Robotergestützte Präparation von Bakterienklonen: Unüberwachte Plasmidisolierung und PCR-Sequenzierung

**Der gemeinsamen Naturwissenschaftlichen Fakultät
der technischen Universität Carolo-Wilhelmina
zu Braunschweig
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)**

**genehmigten
D i s s e r t a t i o n**

**von Dipl.-Biologe Univ. Gerhard Kauer
aus Bamberg**

1. Referent: Prof. Dr. W. Deckwer
2. Referent: Prof. Dr. H. Lehrach
eingereicht am: 14.05.1998
mündliche Prüfung am: 22.01.1999

1999
(Druckjahr)

1 Inhalt

2 Zusammenfassung

Gerhard Kauer

Robotergestützte Präparation von Bakterienklonen:
Unüberwachte Plasmidisolierung und PCR-Sequenzierung.

Moderne Sequenzierprojekte haben die Bestimmung der Basenabfolge kompletter Genome zum Ziel. Wegen der entspechend anfallenden Laborarbeiten ist die Bewältigung derartiger Vorhaben nur noch durch weitgehende Automation der notwendigen Präparationen möglich.
In dieser Arbeit werden kommerziell erhältliche Roboter und apparative Eigenentwicklungen in der Weise kombiniert, daß eine unüberwachte Präparation von Plasmidvektoren aus lebenden Bakterien möglich wird. Die Automatisierung schließt eine PCR-Sequenzierung mit ein, so daß am Prozeßende auftragsfähige Lösungen für Sequencer des Typs ABI bzw. A.L.F.[1] vorliegen.
In einer nicht sensorisch rückgekoppelten Roboteranlage, wie sie in dieser Arbeit verwirklicht wurde, muß eine ausreichende Lagestabilisierung der Einzelkomponenten gewährleistet sein. Bewegte Teile müssen durch besondere Konstruktionen in definierte Positionen gebracht werden, um auftretende Toleranzen auszugleichen. Ein Ergebnis dieser Arbeit ist unter anderem ein mechanisches Korrektursystem, das erst den automatisierten Betrieb der Plasmidpräparation ermöglicht. Die Plasmid-Präparations-protokolle der Fa. Qiagen wurden auf die entstandene Anlage portiert. Hierzu wurde eine Vakuumkammer entwickelt, die über eine eigens hierfür konzipierte elektronische Ansteuerung verfügt. Über diese Elektronik wird unter anderem die Vakuumkammer von einer Regelungs- und Steuerungssoftware angesteuert. Diese Software ist objektorientiert entworfen, gemäß Iso9000 dokumentiert und in C++ implementiert worden. Sie regelt nicht nur die Ventil-einstellungen der Vakuumkammer sondern moderiert auch die Kommunikation der übrigen Roboter und die Ansteuerung der am Ablauf des Protokolls beteiligten Maschinen.

[1] Unter dem „Typ“ A.L.F. fallen auch Protokolle für Sequencer wie z.B. der Firma Licor udgl. Die Einteilung erfolgt gemäß der jeweiligen Farbmarkierungsstrategie für die zu sequenzierenden DNA-Fragmente (Vierfarb- bzw. Einfarbmarkierung).

3 Steigerung des Probendurchsatzes bei Mega- Sequenzierprojekten

Anstrengungen, die komplette Erbinformation ganzer Organismen zu erhalten, nehmen in den vergangenen Jahren in enormem Maß zu. Beginnend mit der Sequenzierung eines Phagengenoms (Bakteriophage T7: 38000 Basenpaare, Bakteriophage λ: 48514 Basenpaare), über das Genom von *Escherichia coli* ($4.6 \cdot 10^6$ Basenpaare) bis hin zur Hefe *Saccharomyces cerevisiae* ($1.3 \cdot 10^7$ Basenpaare) [Row97] als ersten Vertreter der Eukaryonten steigerte sich die Anzahl zu sequenzierender Basenpaare um das fast 600fache. Schon ist das Erbgut des Menschen mit mehr als $3 \cdot 10^9$ Basenpaaren im „Human Genome Project" Ziel dieser Anstrengungen geworden. Im Arabidopsis-Projekt soll das Genom der Nutzpflanze *Arabidopsis thaliana* untersucht werden. Sie ist ein Modellsystem für die Molekularbiologie der Pflanzen [Mey85, Pan87]. *Arabidopsis thaliana* weist fünf Chromosomen auf, von denen bereits detaillierte physikalische Karten für eine effiziente Genisolation bereitstehen. An genetischem Material des Chromosoms Nummer 4 wird in der vorliegenden Arbeit die Funktion der hier vorgestellten automatisierten Präparationstechnik erprobt. Die Entschlüsselung der ca. 17 Megabasen des Chromosoms Nummer 4 von *Arabidopsis thaliana* wird schätzungsweise zu 3200 neu entdeckten, exakt charakterisierten Genen führen.
Die enormen Mengen der zu sequenzierenden DNA-Information kann mit den bisherigen Labormitteln und Laborpersonal kaum bewältigt werden. So sind neue Technologien notwendig, welche den Probendurchsatz in diesen Forschungsvorhaben bei vertretbarem finanziellen Aufwand erheblich erhöhen können. Dabei werden in der laufenden Entwicklung zwei sich gegenseitig beeinflussende Strategien offensichtlich: Die Miniaturisierung der Laborabläufe einerseits und die unüberwachte Automatisierung bewährter Laborprozeduren andererseits. Über eine intensive Nutzung der Automation ist eine Steigerung des Probendurchsatzes durch Parallelisierung dieser Methoden möglich.

3.1 Miniaturisierung von Laborabläufen

3.1.1 Miniaturisierte Elektrophoreseanalysatoren

Die Elektrophorese nutzt die Trennung von Biomolekülen aufgrund ihrer Ladung und Größe aus. Die Moleküle werden einem elektrischen Feld ausgesetzt und wandern durch ein inertes Medium, das die größeren Moleküle sterisch stärker in ihrem Wanderungsverhalten behindert als die kleineren. Folge hiervon ist, daß sich eine Trennung des als Probe aufgetragenen Gemisches ergibt. Häufigstes Trennungsmedium für Biomoleküle ist ein Gel (Agarose, Polyacrylamid).
Wird die Elektrophorese miniaturisiert, so konnte beobachtet werden, daß sich die Trennungsleistung erheblich steigern ließ.

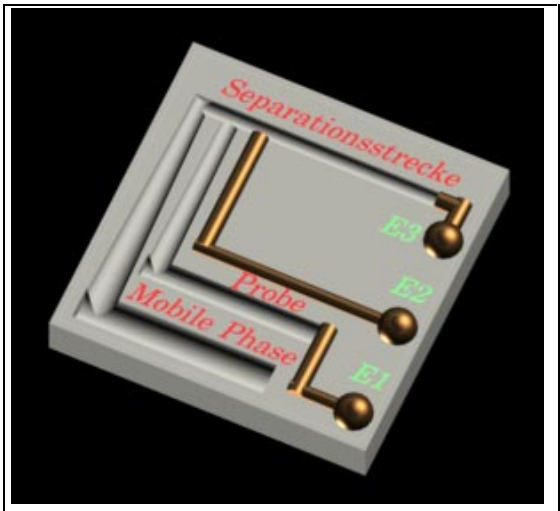


Abbildung 3-1: Elektrophorese-Chip

So können im Miniaturbereich wesentlich höhere Spannungen angelegt werden. [Pac90, Man92]. Die Weiterentwicklung der Mikrostrukturen im Elektrophorese-Chip [Har92, Sei93, Har93] ermöglichte eine weitere Steigerung der Trennungseffizienz. In ein inertes Trägermaterial sind kleine Kanäle eingeätzt. Die für den Betrieb des Elektrophoresechips notwendigen Elektroden und deren Anschlüsse sind ebenfalls fest in das Trägermaterial eingebettet. Durch die besondere Anordnung der Elektroden kann die Probe allein aufgrund ihrer Ladung aus einem Reservoir zur Separationsstrecke transportiert werden [Woo94]. Es sind in diesem Fall keinerlei mikromechanische Vorrichtungen notwendig (vgl. Abbildung 3-1): Nur dadurch, daß zunächst die Elektroden E1 und E2 aktiv sind, wird die Probe an die Trennungsstrecke gebracht. Werden dann die Elektroden E2 und E3 mit 25kV aktiviert, so trennt sich die Probe elektrophoretisch in der Trennungskapillare. Auch die Detektion der Biomoleküle nach der Separation soll in den Chip integriert werden [Ver91]. Am MIT Lincoln Laboratory/Whitehed Institute wird ein Elektrophrese-chip entwickelt, auf dessen Fläche von nur 6.5 x 4.5 cm mehrere hundert Kanäle aus 40 µm breiten Kapillaren geätztwurden (vgl. Abbildung 3-2). Mit diesem Gerät soll in hohem Ausmaß Kapillar-

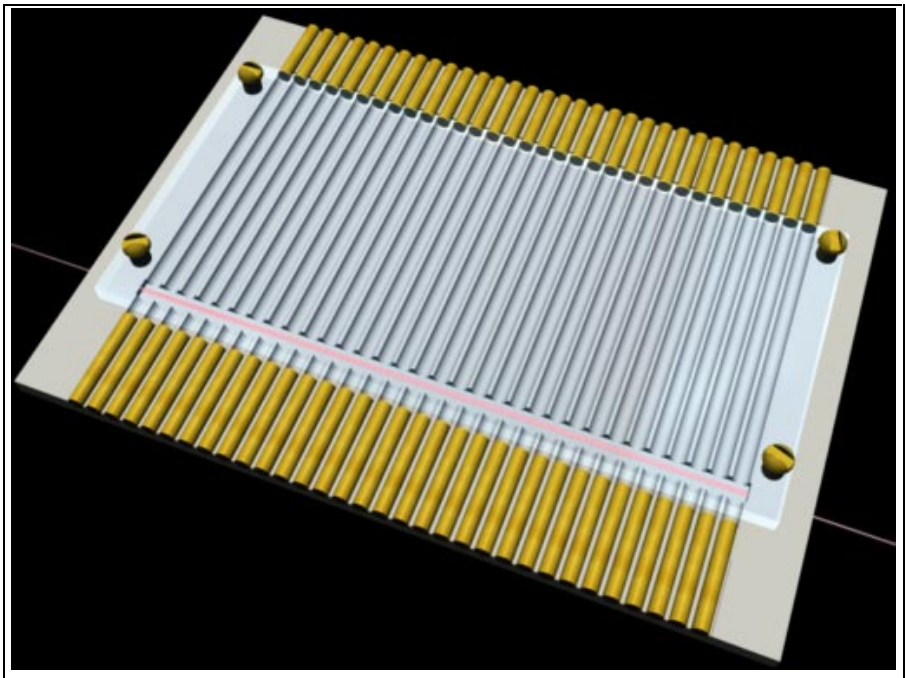


Abbildung 3-2: Modellzeichnung eines Elektrophoresechips für hochparallele Elektrophorese. Die geätzten Kapillaren sind mit einem Glasplättchen abgedeckt. Ein Laserstrahl (rot gezeichnet) dient zur Fluoreszenzanregung farbmarkierter (DNA-) Moleküle. Der Chip hat eine Abmessung von ca. 2.5cm in seiner Diagonalen.

lektrophorese betrieben werden. Das Ziel dieser Entwicklung soll ein kontinuierlicher Probendurchsatz mit 500 bis 1000 Spuren pro Stunde sein, wobei in jeder dieser Spuren eine 300 Basenpaar lange DNA-Probe sequenziert werden kann. Dieser Durchsatz steigert die Sequenzierkapazität im Vergleich zu aktuellen Methoden um das Doppelte bis Dreifache. Durch die aus der Mikrochip-Technologie entliehene Verfahrenstechnik zur Herstellung von mikroskopisch kleinen Strukturen aus Wafern, ist die industrielle Produktion solcher Elektrophorese-Analysatoren sehr ökonomisch und auf Massenproduktion ausgelegt. Die elektrophoretische Auftrennung erfolgt entlang der in das Trägermaterial eingeätzten Kapillaren. Zur Detektion der getrennten DNA Fragmente sind diese fluoreszenzmarkiert. Am Ende der Kapillare werden die Fragmente an einem Laser vorbeigeführt, so daß sich dort ein Meßsignal über die auftretende Fluoreszenz ergibt. Dieses De-tektionsverfahren ist Standard in den herkömmlichen Analysatoren wie A.L.F (Pharmacia) oder dem ABI-Sequencer. Für die Probenauftragung auf die miniaturisierten Elektrophorese-Analysatoren ist der Einsatz neuer Technologien erforderlich, weil in den winzigen Kapillaren ebenso winzige Probenvolumina bewegt werden müssen. So können im Mikrometerbereich Pumpen, Ventile und Motoren hergestellt werden [Pet82, Tav90, Wis91].

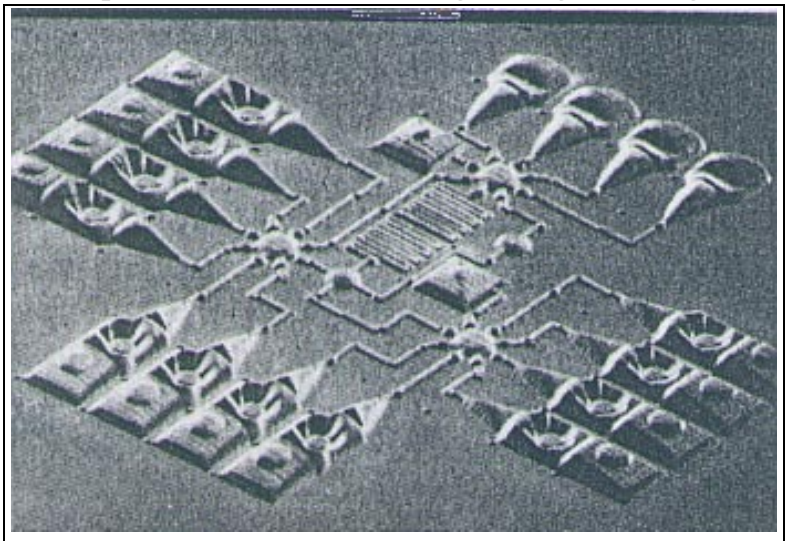


Abbildung 3-3: Mikrofluß-Struktur, die über eine 3D-Laserführung aus einem Silikonträger geätzt wurde. Der Doppelpfeil ist ca. 100µm lang.

Am MIT Lincoln Laboratory/Whitehead Institute plant man, zur Integration in einen Elektrophorese-Chip dreidimensional geätzte Mikrostrukturen [Blo92] einzusetzen. Bei diesem Verfahren wird die zu erzeugende Struktur zunächst per Computer dreidimensional gezeichnet und anschließend durch einen akkustooptisch bewegten Argon-Laserstrahl aus einem Polymer gewonnen. Diese Technik ermöglicht es auch, die Mikrokapillaren noch feiner und in dreidimensional exakt definierbarer Form aus dem Trägermaterial herauszuarbeiten. Auf diese Weise ist es beispielsweise möglich, die Trennstrecke einer Kapillare erheblich zu verlängern (vgl. Abbildung 3-3) indem diese in schmale Serpentinen gelegt wird.

3.1.2 Miniaturisierte PCR-Maschinen

Bei der Polymerase-Kettenreaktion (Polymerase Chain Reaction=PCR) werden bis zu 6 Kilobasen große DNA-Fragmente dadurch vervielfacht, daß die DNA-Probe zunächst thermisch bis zu ihrer Schmelztemperatur erhitzt wird, wodurch sich die helikalen Fragmente in ihre Einzelstränge trennen. Im Reaktionsgemisch befinden sich des weiteren eine thermisch stabile Polymerase (z.B. aus ***T**hermo-philus **aq**uaticus* gewonnen) und zwei Primer, wodurch die während des Schmelzens entstandenen DNA-Einzelstränge beim nachfolgenden Abkühlen repliziert werden. Die beiden Primer sind dabei so beschaffen, daß sie sich an den zu amplifizierenden Enden der DNA (jeweils ein Primer für das 3'-und 5'-Ende) anlagern und der Polymerase ein freies 3'-OH-Ende auf dem zu erzeugenden Komplementärstrang anbieten. Durch Verwendung der thermisch stabilen **Taq**-Polymerase ist es möglich, die Reaktion zyklisch so lange zu wiederholen, bis eine ausreichende Menge replizierter DNA vorliegt. Schon nach 25 Zyklen ist die Ausgangsmenge an DNA um den Faktor 10^6 vermehrt. Die Replikation ist durch die Auswahl der Primer sehr spezifisch.

Am Lawrence Livermore National Laboratory, USA wurde bereits in den Jahren 1993/94 ein Prototyp einer miniaturisierten PCR-Maschine entwickelt [Sti94, Nor93]. Auf einem Silicon-Waver von ca. 10 cm Durchmesser wurden winzige Vertiefungen eingeätzt, in denen die beschriebene PCR ablief. Die Reaktionskammer ist nach oben hin durch eine Glasplatte abgeschlossen. Die für die Reaktion notwendigen Nucleotide wurden während der Zyklen von außen in die nur 5mm²x0.5mm große Reaktionskammer des Chips gepumpt. Wegen der geringeren Abmessungen der Silicon-Basis konnten die Temperatursprünge wesentlich schneller eingestellt werden, als bei herkömmlichen PCR-Maschinen. Bereits mit dem beschriebenen Prototyp konnte auf diese Weise die gesamte PCR 4 mal schneller vollzogen werden.

3.1.3 Miniaturisierte Proben-Arrays

Besonders interessant für DNA-diagnostische Zwecke ist die miniaturisierte Anordnung immobilisierter DNA-Oligomere. Auf dem Chip werden an genau definierten Stellen n DNA-Oligomere bekannter Sequenzen gekoppelt, die als Sensormoleküle fungieren. Somit ist sowohl die Position als auch die Sequenz der Oligomere auf dem Chip festgelegt und per Computer erfaßbar. Der komplette Satz für DNA-Oktamere ist z.B. 4^8=65536 Oligonucleotide. Derartig ausgestattet könnte selbst eine Sequenzierung durch Hybridisierung stattfinden [Bai88, Drm89, Khr89, Khr91, Sou92].

Probleme, die sich dadurch ergeben, daß Oligonucleotide in Abhängigkeit ihrer Sequenz eine unterschiedliche Bindungsstärke (Disso-ziationstemperatur) aufweisen, können durch ein patentiertes Verfahren zur Nivellierung der Bindungsstabilitäten ausgeglichen werden [Hoh96].

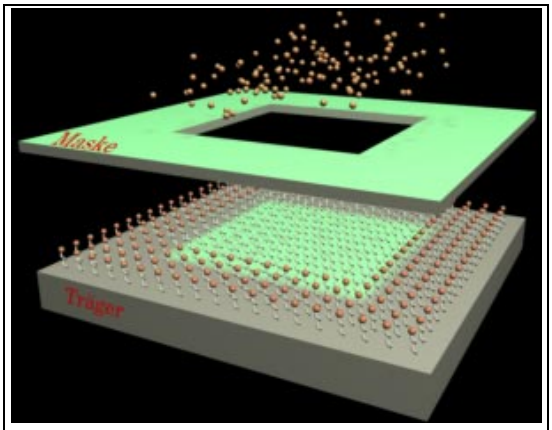


Abbildung 3-4: Die Schutzgruppen der belichteten Molekel diffundieren ab.

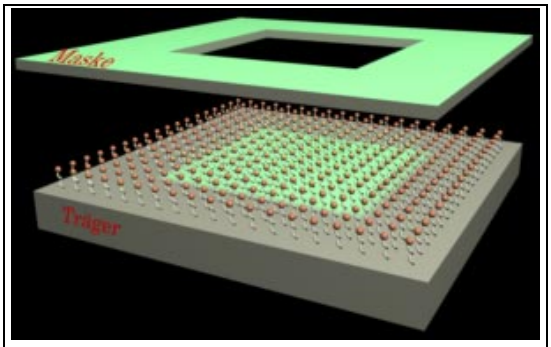


Abbildung 3-5: Fotolabile Schutzgruppen (rote Kugeln), die über Spacer- moleküle am Träger gebunden sind, werden selektiv belichtet (grünes Licht).

Hybridisierungsexperimente mit kurzen, bis zu 14 Basen langen Oligonucleotiden wurden schon erfolgreich zur Kartierung von Klonbibliotheken eingesetzt [Cra90, Hoh93], aber auch die Sequenzbestimmung kurzer DNA-Moleküle ist bereits gelungen [Drm93], wobei allerdings die Oligonucleotide einzeln hybridisiert wurden und die DNA-Fragmente auf einem Träger fixiert vorlagen. Arbeiten von [Fod91, Sou92, Pea94, Mir94, Mat95] beschäftigten sich mit der Synthese von Oligomerrastern mit einer Vielzahl von Oligonucleotiden.

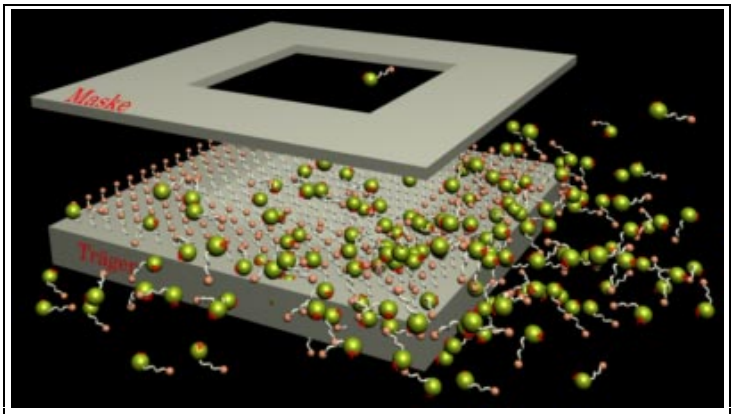


Abbildung 3-6: Auf chemischem Weg werden z.B. „A“-Nucleotide (grüne Kugeln, trans geschützt) an die nun freien Bindungsstellen gebunden.

Die Einsetzbarkeit eines high-density Arrays für diagnostische Zwecke unter Verwendung von 15meren wird von [Fod93] und [Dut94] erwähnt. Die Synthese großer Oligomerraster auf kleinen Oberflächen beschreibt [Mas92] und [Chr96]. Ist die zu erwartende Zielsequenz, z.B. in der Diagnostik bekannt, so kann der Chip mit einer entsprechend geringeren Anzahl an Sensormolekülen ausgerüstet sein [Bai93]. Eine unbekannte DNA-Probe wird auf diesem Chip zur Hybridisierung aufgetragen und das Hybridisierungssignal über hochauflösende CCD-Kameras und Methoden der Fluoreszenzmarkierung direkt am Chip gemessen. Die Grundlage zur Herstellung von Chips mit immobilisierten Oligonucleotiden definierter Sequenz ist ein photolithographisches Verfahren.

Auf einem Trägermaterial bieten Linkermoleküle mögliche Anbindungsstellen für Nucleotide (vgl. Abbildung 3-5). Die Linkermoleküle sind mit Schutzgruppen versehen, die auf fotochemischem Weg abgespalten werden können (vgl. Abbildung 3-4). Ist die Schutzgruppe unter Lichteinwirkung entfernt, so kann auf chemischem Weg ein Nucleotid an das Linkermolekül gebunden werden. Das auf diese Weise gebundene Nucleotid ist ebenfalls mit einer fotolabilen Schutzgruppe versehen (vgl. Abbildung 3-6). Um nun gezielt bestimmte Oligonucleotidsequenzen auf dem Chip zu erhalten bewirken geeignete Masken eine selektive Belichtung des Trägermaterials, so daß nur bestimmte Bereiche für die nächste Kopplungsreaktion dem Licht ausgesetzt werden.
Auf diese Weise erhält man die gewünschte Sequenz (vgl. Abbildung 3-7), indem einem belichteten Abschnitt mit den entschützten Kopplungsgruppen Nucleotide eines bestimmten Typs zur Reaktion angeboten werden.

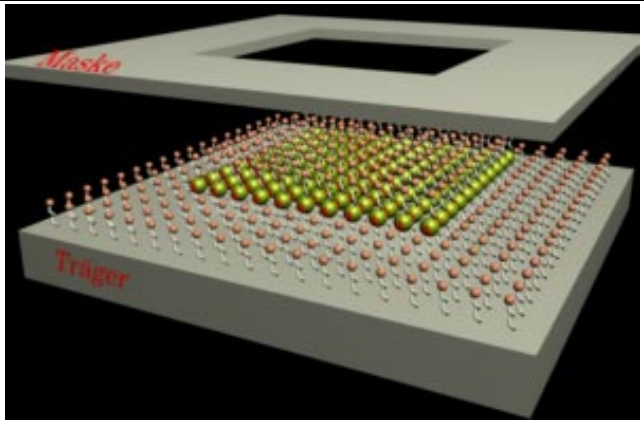


Abbildung 3-7: Die nun gebundenen „A“ Nucleotide weisen wiederum fotolabile Schutzgruppen auf. Der Vorgang (ab Abbildung 3.4) kann erneut beginnen.

Das Anwendungsgebiet solcher Chips ist sehr weit gesteckt: Identifizierung von Erbkrankheiten, schnelle Klassifizierung von Krankheitserregern (z.B. Salmonellentypisierung), Identifizierung von Krebsgenen, schnelle Testverfahren auf die krebserregende Wirkung von Stoffen, Qualitätskontrolle während eines Produktionsvorganges durch fermentierende Organismen (Bierhefe, Käsekulturen), Expressionsstudien und Evolutionsforschung. Die Firma Nanogen entwickelte einen Chip, bei dem die zur Hybridisierung aufgetragene DNA über eine Ladung an die interessierende Reaktionsstruktur im Chip herangeführt wird. Auf diese Weise kann die Effizienz der Hybridisierung gesteigert werden [Nan95].

3.1.4 Miniaturisierte Detektionssysteme

Moderne CCD-Kameras mit Peltierelement gekühltem „**C**harge **C**oupled **D**evice“ sind schon empfindlich genug, um z.B. auch schwache Fluoreszenzsignale zu detektieren. Somit sind diese Geräte auch im Bereich der Gelelektrophorese einsetzbar, sofern die Kontrastunterschiede der aufgenommenen Strukturen nicht zu groß sind. Die aufgenommenen Bilder können digital ausgewertet werden [Mac89, Yam91, Pen92, Jac88, Cha91, Gre92, Che90, Swe91,Kos92, Kar91].
Allerdings reicht die Auflösung dieser Systeme bei Weitem nicht aus, wenn eine signifikante Signaldetektion auf den oft nur 1-2 cm² kleinen Chips erfolgen soll. Ein hierfür geeigneter, hoch auflösender, äußerst empfindlicher Detektor ist bei

[Egg94] beschrieben. Im Prinzip werden bei diesem System die biologischen Proben direkt auf den lichtempfindlichen Elementen („Pixel“= Picture Element) eines CCD-Chips immobilisiert. Die Probenmoleküle können beispielsweise über Chemolumineszenz oder Fluoreszenz detektierbar sein, wobei an die zu untersuchende Reaktion eine zeitabhängige Zunahme oder Abnahme der emittierten Photonen gebunden ist. Findet die Reaktion statt, so werden die Photonen in unmittelbarer Nachbarschaft der lichtempfindlichen Elemente des CCD’s emittiert. Der Detektor ist dadurch etwa 10^3 bis 10^4 mal empfindlicher als ein fotografischer Film, bzw. 10 bis 100 fach empfindlicher als ein Phosphorimager.
Der bei [Egg94] realisierte Chip hat eine fotosensitive Fläche von 420x420 lichtempfindlichen Elementen. Auf jedem dieser „Pixel“ ist DNA immobilisiert. Die Auslesezeit eines derartig modifizierten CCD-Chips beträgt nur etwa eine Sekunde, wodurch sich auch Kinetiken entsprechender Hybridisierungsreaktionen untersuchen lassen. Gegenwärtig wird an wiederverwendbaren CCD-Chips gearbeitet, bei denen die immobilisierten Proben nicht unmittelbar auf dem Chip aufgetragen werden, sondern sich auf transparenten Trägermaterialien in unmittelbarer Nähe zum CCD befinden.

3.1.5 Miniaturisierte Labors für diagnostische Methoden

Die beschriebenen miniaturisierten Elemente können zu größeren Einheiten zusammengefaßt werden. So könnte die Integration eines Elektrophoresechips mit einer geeigneten Detektionsstruktur zu einer kompletten Laboreinheit führen, die z.B. zur miniaturisierten Sequenzierung herangezogen werden könnte. Schaltet man dieser Einheit noch eine PCR-Stufe vor, so wäre eine PCR-Sequenzierung mit nachfolgender Signaldetektion auf einem einzigen Chip möglich.
An der University of Michigan [Bur95] wird zur Zeit an der Integration eines DNA-Analysesystems gearbeitet, das den Ort der Probenauftragung, die notwendigen Reaktionskammern, sowie die Elektrophorese und Detektion auf einen Chip zum Ziel hat und im Human Genome Project eingesetzt werden soll.

3.2 Roboter zur Unterstützung von Routinemethoden

Um in einem Labor automatisierte Verfahren zu etablieren ist es nicht zwingend notwendig, die Routineverfahren zu miniaturisieren. Es ist ebenfalls möglich ein Robotersystem zu konzipieren, das die manuellen Tätigkeiten des Menschen ganz oder teilweise ersetzt, um letztendlich eine Steigerung des Probendurchsatzes zu erreichen. Sind die oben beschriebenen miniaturisierten Verfahren verfügbar, so können die betroffenen Präparationsschritte sukzessive gegen die neuen Methoden und Geräte ausgetauscht werden.

Folgende manuelle Tätigkeiten sind typisch für eine Laborpräparation (mit besonderem Schwerpunkt auf Plasmidpräparation als Vorbereitung zur PCR- Sequenzierung) und müssen durch geeignete Roboter unterstützt werden:

- Pipettieren
- Transport von Verbrauchsmaterial und Chemikalien
- Durchsaugen von Flüssigkeiten durch Filter
- PCR-Reaktion

3.2.1 Pipettierroboter

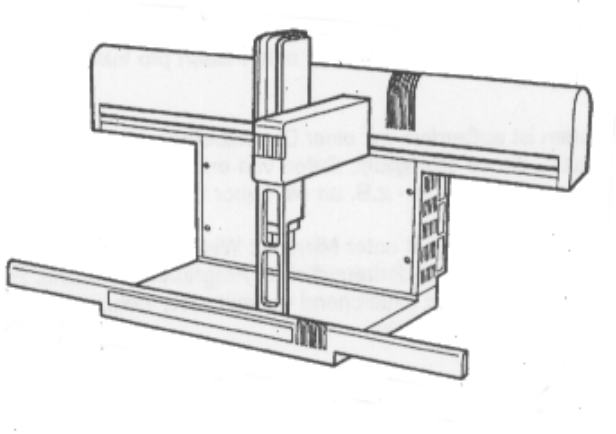


Abbildung 3-8: Biomek 2000 Pipettierroboter.

Moderne Pipettierroboter lokalisieren auf einer Arbeitsfläche normiertes Labormaterial an definierten Positionen und ermöglichen so die Installation von erprobten Laborprotokollen. Beispielsweise können auf diese Maschinen über Hilfsroboter Mikrotiterplatten, Pipettenspitzen oder Reservoirs für Pufferlösungen usw. eingestellt, bzw. nach Gebrauch wieder aus dieser Arbeitsstation entfernt werden. So können alle notwendigen Pipettierschritte zur Vorbereitung einer PCR oder einer Plasmidpräparation abgearbeitet werden, um dann das Produkt dieser Pipettierung zur weiteren Präparation über den Greifroboter in die zugehörige Maschine einzustellen. Der in der vorliegenden Arbeit verwendete Pipettierroboter ist der Biomek 2000 (Fa. Beckman, vgl. Abbildung 3-8). Er besteht aus einem dreidimensional programmierbaren Multifunktionsarm, der neben einer Pipettenaufnahme (Single- und 8-Kanal Pipetten) auch andere Werkzeuge (Whasher, Kolonienstempel) einsetzen kann. Die Maschine verfügt über einen eigenen internen Rechner, der die Prozessierung der vorher festgelegten Protokolle übernimmt. Der Rechner wird über eine serielle Schnittstelle mit einem zentralen Steuerungsrechner verbunden. Über diese Leitung wird ein vorher festgelegtes Arbeitsprotokoll übergeben, welche alle Angaben zu den notwendigen Pipettier- und Arbeitsschritten enthält. Es besteht bei diesem Roboter allerdings noch die Möglichkeit über eine weitere serielle Schnittstelle (Terminal-Emulation) auf Hardwareebene mit dem internen Steuerungsrechner des Biomek 2000 zu kommunizieren, wodurch die „tiefen“ Funktionen der Maschine, die normalerweise verborgen sind, zugänglich werden. Somit kann dieser Pipettierro-

boter an viele Schwierigkeiten der Protokollimplementation über eine geeignete Steuerungssoftware angepaßt werden, die zu diesem Zweck allerdings eigens geschrieben werden muß (z.B. in C++), weil die derzeitigen Versionen der kommerziell erhältlichen Software die notwendigen Operationen bei weitem nicht beherrschen. Das Betriebssystem des Biomek 2000 erlaubt es, fremde Steuerungssoftware, die für andere Roboter verantwortlich ist, zu starten und unterstützt in Verbindung damit eine einfache Interprozeßkommunikation.

3.2.2 Greifroboter

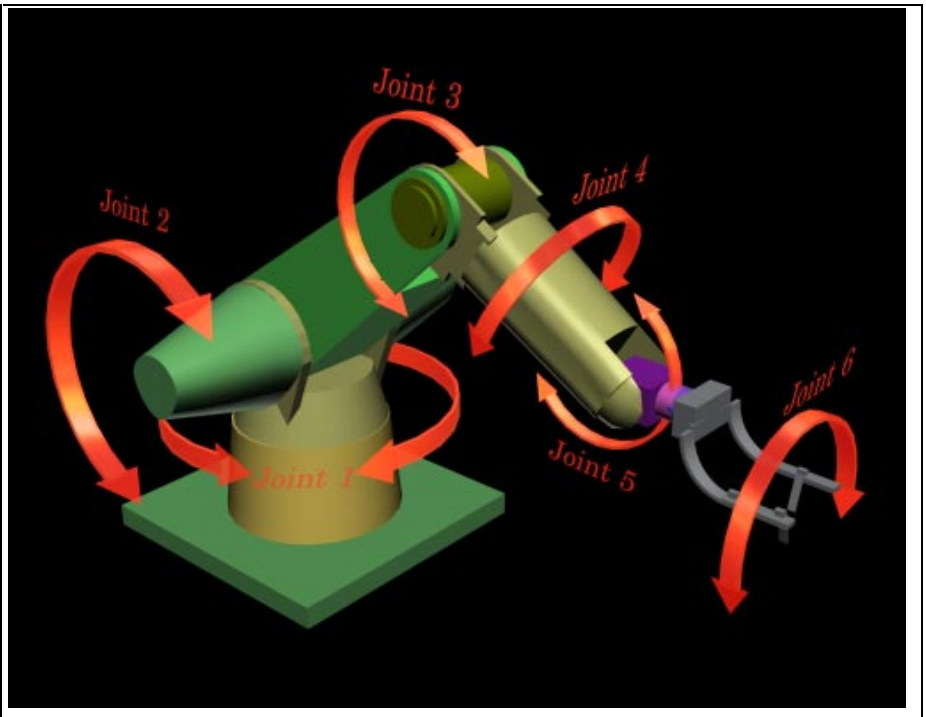


Abbildung 3-9: Der Industrieroboter „CRS 465“ (CRS Robotics, Kanada) verfügt über sechs Freiheitsgrade. Die motorgetriebenen Achsen sind im „Joint-Modus“ über eine RISC-Workstation einzeln oder kombiniert ansteuerbar.

Greifroboter zeichnen sich dadurch aus, daß sie über eine handähnliche Vorrichtung am Ende eines allseits beweglichen Armes verfügen. Sie nehmen die Verbrauchsmaterialien auf und bewegen sie an exakt definierte Stellen im System. In der vorliegenden Arbeit wurde ein CRS 465 (Industrieroboter) eingesetzt, der über sechs Freiheitsgrade verfügt (vgl. Abbildung 3-9). Schrittmotoren, die sich in den sechs Gelenken des Roboters befinden, ermöglichen Drehbewegungen mit einer Genauigkeit von ±0,05 mm. Angesteuert werden die Motore über eine eigene externe C500 RISC Workstation, die ihrerseits über eine serielle Schnittstelle mit dem Steuerungsrechner kommuniziert. Die „Hand“ des Roboters wurde für die speziellen Anforderungen ebenfalls in der vorliegenden Arbeit konstruiert und gebaut. Die Risc-Workstation (C500) ist für die Berechnungen der im Bewe-

gungsablauf aktuellen Motorstellungen verantwortlich. Die Ansteuerung des C500 erfolgt durch eine eigene maschinennahe Programmiersprache, die vom Hersteller des CRS 465 entwickelt wurde. RAPLII (**R**obot **A**utomation **P**rogramming **L**anguage, Generation **II**) [CRS94]. Es handelt sich dabei um eine zeilenorientierte Programmiersprache, die in eingeschränktem Umfang auch modulares Programmieren unterstützt.
Der CRS465 erlaubt die Ansteuerung in 4 verschiedenen Modi, die sich durch die unterschiedliche Interpretation seiner Freiheitsgrade ergeben.

- Joint Mode
- World Mode
- Tool Mode
- Cylindrical Mode

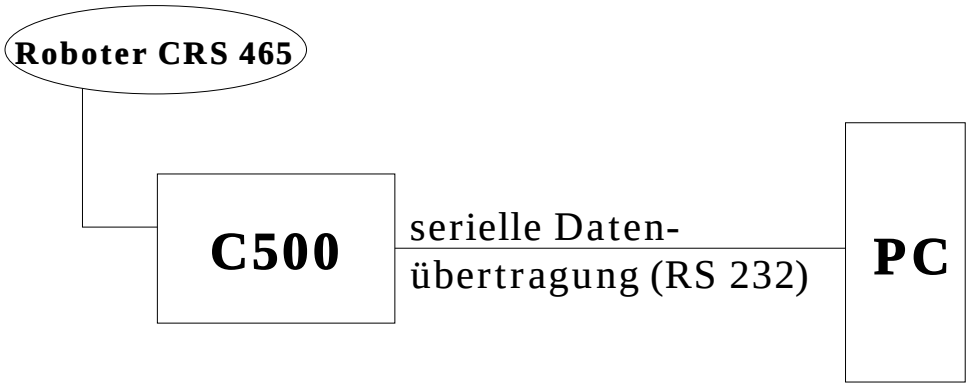


Abbildung 3-10: Ansteuerung des CRS 465 über einen PC.

Im Joint-Modus wird jede Achse einzeln angesteuert. Man kann den Roboter somit an jeder seiner Achsen einfache Rotationen durchführen lassen (vgl. Abbildung 3-9).
Im World-Modus spannt sich ein kartesisches Koordinatensystem um den Roboter, das seinen Ursprung im Zentrum seiner Basis hat (vgl.Abbildung 3-11).

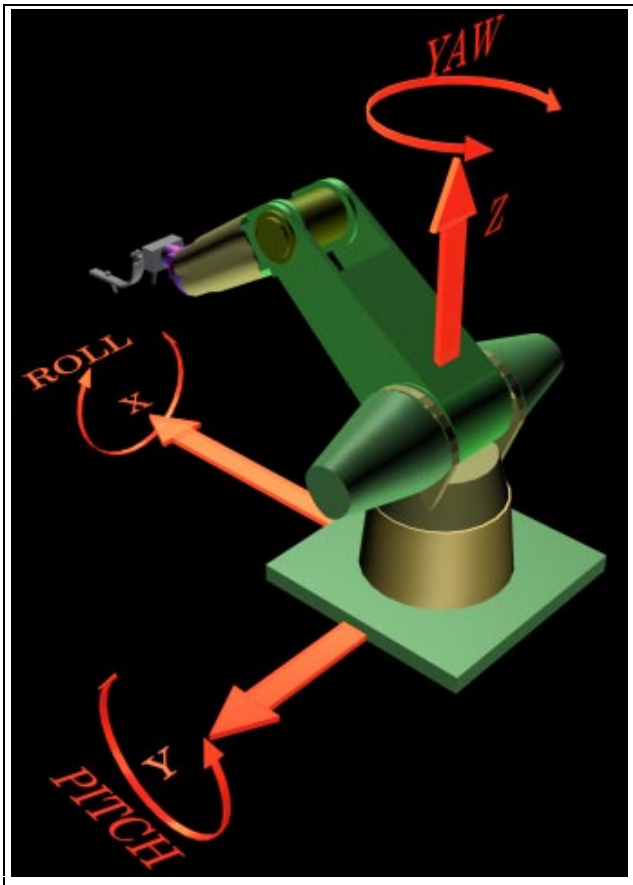


Abbildung 3-11: Im „World Modus“ des CRS 465 spannt sich ein kartesisches Koordinatensystem um den Roboter. Zusätzlich sind noch Rollbewegungen des Greifers längs der Achsen möglich.

Dem Roboter können in diesem Modus absolute Koordinaten vorgegeben werden, die dann von der Maschine angefahren werden. Wird beispielsweise eine Tangentialbewegung parallel zu einer Oberfläche programmiert, so berechnet die C500 zu jedem Zeitpunkt innerhalb der noch nicht zu Ende gebrachten Bewegung des Armes zur Zielstellung hin, alle zugehörigen Motordrehstellungen, damit der Roboterarm sich in der geforderten Weise parallel zur Oberfläche bewegen kann. Allerdings sind dieser Bewegungsform Grenzen hinsichtlich Reichweite und Verfahrweg auferlegt. Eine Neigung des Greifers ist zusätzlich über die Einstellungen „Yaw“, „Pitch“ und „Roll“ möglich (vgl. Abbildung 3-11).

Der Tool-Modus unterscheidet sich kaum vom beschriebenen World-Modus. Allerdings ist der Ursprung nun nicht die Basis des Roboters sondern seine Werkzeugaufnahme (Greifer).
Die C500 erlaubt es, einen Offset zu definieren, um den Ursprung z.B. an einen beliebigen geometrischen Mittelpunkt des Greifers zu verschieben (vgl. Abbildung 3-13). Mit dem Cylinder-Modus kann man den CRS465 in einem Polarkoordinaten-system ansteuern (vgl. Abbildung 3-12). Anstelle der Welt-Koordinaten X und Y werden die Polar-koordinaten δ und R verwendet. δ ist der Winkel, welcher entgegen dem Uhrzeigersinn über die z-Achse gemessen wird, wenn von der x-Achse ausgegangen wird. R ist dabei die Position vom Ursprung in Richtung δ. Ein Teach Pendant erlaubt die manuelle Definition der Koordinatenendpunkte. Die Abfrage der Punktdaten aus dem C500 ist über eine Interprozeßkommunikation möglich. Ein DDE-Server steuert die intermaschinelle Regulation, stellt also die notwendigen Kommunikationsprotokolle bereit, mit der RAPLII-Programme aufgerufen werden können. Diese RAPLII-Programme werden permanent in der C500 abrufbar gespeichert.

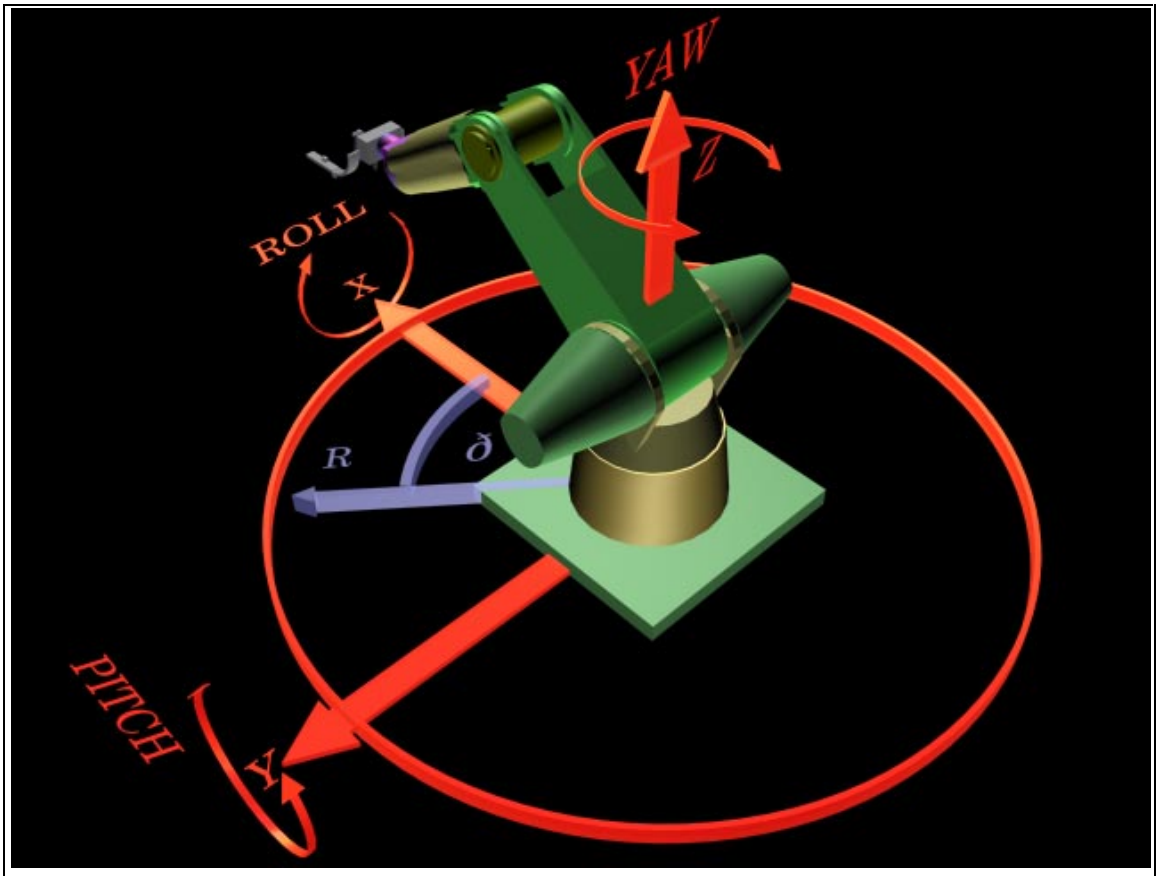


Abbildung 3-12: Cylinder-Modus des CRS 465. Hier können Polarkoordinaten in der X-Y Ebene angegeben werden.

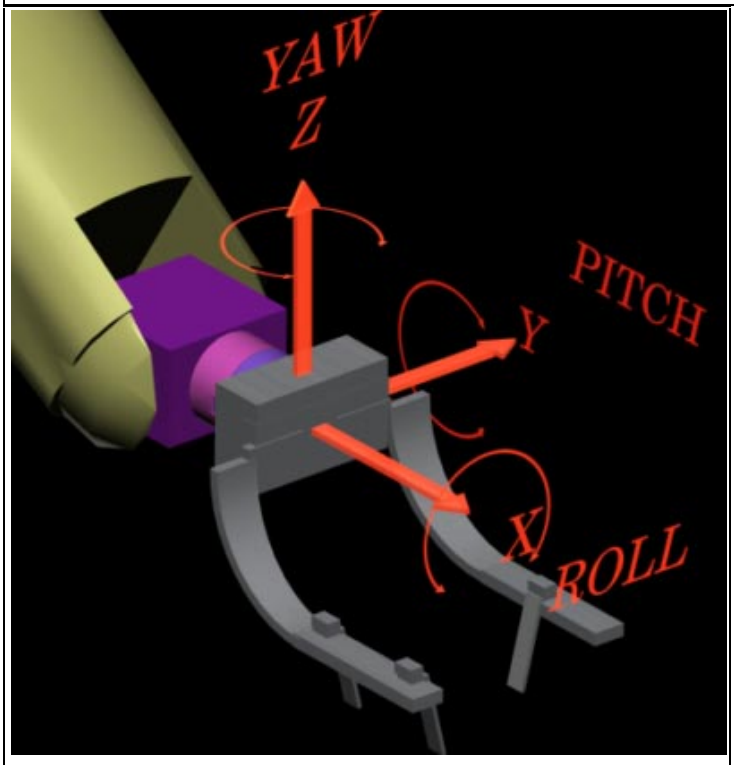


Abbildung 3-13: Tool Modus. Das Koordinatensystem hat seinen Ursprung am Greifer.

3.2.3 PCR-Roboter

Die Polymerase-Kettenreaktion (PCR) vervielfacht ein DNA-Segment, wenn es zwischen zwei definierten Primerstellen eingeschlossen ist [Mul87,Sai88]. Werden gleiche Mengen an Primern eingesetzt, dann entstehen über die PCR doppelsträngige DNA-Kopien, wird hingegen ein Primer im Überschuß eingesetzt, dann erhält man diesem Überschuß entsprechend einzelsträngige Kopien der vervielfältigten DNA. Sowohl einzelsträngige als auch doppelsträngige DNA kann zur Sequenzierung eingesetzt werden [Eng88, Gyl88, Inn88].
In sequenzierintensiven Projekten werden die zu analysierenden DNA-Fragmente in Plasmide kloniert, die dann zunächst in einem auf Agar wachsenden, definiertem Raster von Bakterienkolonien (z.B. *Escherichia coli* XL1 Blue) vorliegen. Auch das anschließende Aufnehmen der Kolonien aus diesem Raster in Kulturröhrchen kann über eine visuelle Rückkoppelung (Bildanalyse) automatisiert werden [Leh94]. Die lebenden Bakterienklone liefern dann über eine ca. 12 Stunden dauernde Inkubationszeit (37°C) genügend Material, um in einer Präparation die zur Sequenzierung notwendigen Plasmidkopien zu gewinnen. Ist die Konzentration der gewonnenen Plasmide zu gering, so kann über eine PCR deren Kopienzahl nachträglich erhöht werden.

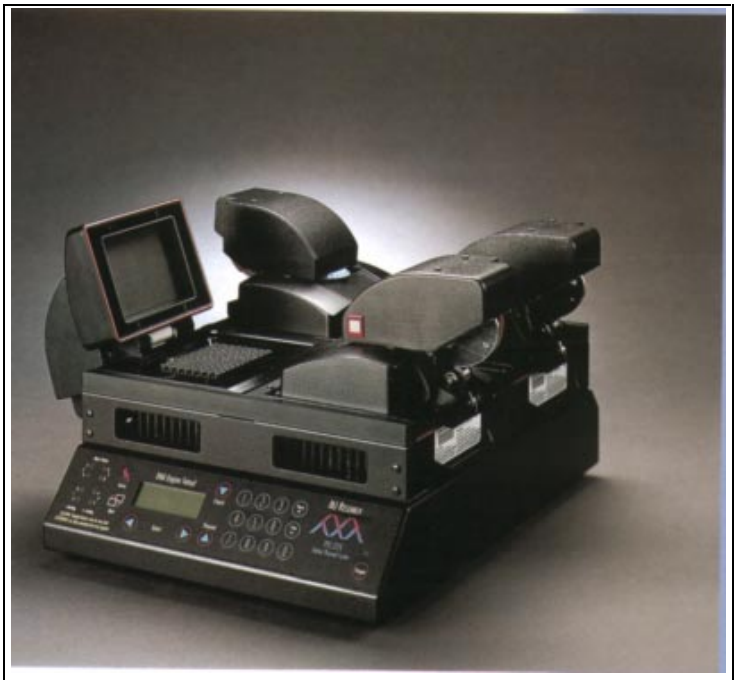


Abbildung 3-14: PCR-Maschine von MJ-Research mit automatisch schließenden Deckeln (Modell PCT-225)

Ein gängiges Laborverfahren ist die PCR-Sequenzierung, bei der die PCR unter Kombination mit Didesoxynucleotiden zur Generierung einer geeigneten Sequenzierauftragslösung verwendet wird. Der zentrale Mechanismus der PCR nutzt das Schmelzverhalten der DNA aus. Oberhalb einer bestimmten Temperatur trennen sich die beiden Stränge der DNA. Rasche Abkühlung bewirkt, daß die Einzelstränge kaum wieder renaturieren können. Die im Reaktionsgemisch angebotenen Primer haben nun die Möglichkeit sich an ihre komplementären Stellen der DNA zu binden. Die temperaturstabile Polymerase (Taq aus ***T**hermus **aq**uaticus* bzw. eine

modifizierte T7 Polymerase [Koo93]) kann ihre Polymerisationstätigkeit aufnehmen, und die DNA-Einzelstränge werden wieder zu Doppelsträngen ergänzt. Liegen nun im Reaktionsgemisch Didesoxynucleotide vor, so erfolgt ein Abbruch der Polymerase-Strangregene-ration an statistisch über den Matrizenstrang verteilten Stellen. Eine erneute Hitzedenaturierung der DNA beginnt einen zweiten Zyklus, usw. In Folge entstehen so über die statistischen Strangabbrüche der Polymerasereaktion DNA-Fragmente, deren Länge sich um jeweils ein Nucleotid unterscheiden, was bei der anschließenden Elektrophorese zur Sequenzgewinnung genutzt wird. Für den technischen Aufbau einer automatischen PCR-Anlage ist es somit essentiell, eine rasche Erhitzung und ebenso rasche Abkühlung der Reaktionslösung vorzunehmen. Die technische Lösung für eine derartige Anlage wird durch Peltierelemente angeboten. Der Peltier-Effekt geht darauf zurück, daß in einem aus verschiedenen Metallen zusammengesetzten Stromkreis die an den Lötstellen entwickelte Wärme nicht dem Jouleschen Gesetz entspricht, sondern an der einen um einen bestimmten Betrag höher, an der anderen hingegen um den gleichen Betrag niedriger ist. Sind die beiden Metalle z.B. Wismut und Antimon, so erwärmt sich die Lötstelle beider Metalle, wenn die Elektronen vom Wismut zum Antimon fließen. Die Lötstelle kühlt ab, wenn die Elektronen vom Antimon zum Wismut fließen. Der Thermostrom fließt also immer entgegengesetzt zum Elektronenstrom [Gri77].

Die in dieser Arbeit angesteuerte Anlage verfügt über vier hocheffiziente Peltierelemente, die vier unabhängige PCR's in vier Mikrotiterplatten mit jeweils 96 Reaktionskammern bewerkstelligen können. Damit auch kleine Probenvolumina präpariert werden können, ist es notwendig, daß die Anlage über beheizbare Deckel verfügt. Somit wird die Verdunstungsgefahr während der DNA-Denaturierung vermindert. Während die Proben abgekühlt werden, ist die Kondensation in die Reaktionskammern hinein gerichtet. Damit eine unüberwachte Präparation mit der vorliegenden PCR-Anlage bewerkstelligt werden kann, sind die vier Deckel des Gerätes über ein digitales Interface motorisch ansteuerbar. In dieser Arbeit wurde eine PCR-Maschine von MJ Research eingesetzt (vgl. Abbildung 3-14). Die Ansteuerung erfolgt mittels eines Kommunikationsprotokolls, das mit einer in dieser Arbeit entwickelten Steuerungssoftware über die serielle Schnittstelle (RS232) abläuft. Die im Ergebnisteil beschriebene Ansteuerungssoftware nutzt diese Schnittstelle sowohl zur Einstellung entsprechender PCR-Programme, die auf dem Prozessor der Maschine ausgeführt werden, als auch zur Überwachung der laufenden Ist- und Sollwerte.

4 Grundlegende Elemente des OOD

Wie stabil oder robust ein Gegenstand bei der Ausführung seiner Funktion ist, hängt von der Sorgfalt ab, mit der dieser Gegenstand geschaffen wurde. Dies schließt sowohl seine Planung, als auch die konkrete Umsetzung seines Entwurfs ein. Der Begriff „Gegenstand“ soll dabei in seiner weitesten Form verstanden werden. So kann Beispielsweise auch ein Vorgang Gegenstand einer Problemstellung sein. Schon rein intuitiv kann man abschätzen, daß komplizierte, aus vielen empfindlichen Einzelheiten bestehende Gegenstände eine wesentlich höhere Sorgfalt erfordern, um letztendlich stabil oder robust realisiert zu werden. Ab einer gewissen Komplexität ist es dem menschlichen Geist nicht mehr möglich, einen stabilen Gegenstand zu schaffen, da nicht mehr alle Eventualitäten (z.B. mögliche Situationen für Fehlfunktionen) überschaubar sind. Man kann in einem solchen Fall nur die Reduktion der Komplexität im Einzelnen anstreben, indem beispielsweise die Gesamtheit in kleinere Problemfragmente geteilt wird, die jeweils für sich lösbar sind. Die Gesamtkomplexität steigt dadurch allerdings erneut an, weil die Teilbereiche so geschaffen werden müssen, daß sie über ihre jeweiligen Grenzen hinaus interagieren können („Schnittstelle“) und den angestrebten funktionalen Gegenstand bilden. So muß die Stabilität sowohl der Fragmente als auch der Schnittstellen gewährleistet sein.
Komplexe Software ist ein solcher Gegenstand, der durch die beschriebene Strategie geschaffen werden kann. Der im Folgenden beschriebene objektorientierte Ansatz ist allerdings nicht nur auf die Problemlösung von Softwarefragestellungen anwendbar, sondern ist ein generelles Prinzip zur Division komplexer Systeme in überschaubare und interagierende Untereinheiten. Es handelt sich dabei um eine „natürliche“ Form des Problemverstehens, weil sich dieser Ansatz an der Gegenständlichkeit, dem „Objekt“ der Problemstellung, orientiert. Nachdem der Gegenstand unter objektorientierten Gesichtspunkten modelliert wurde, schließt sich die Implementation in eine konkrete Programmiersprache an. Je mehr Elemente des objektorientierten Denkansatzes von dieser Sprache unterstützt werden, desto einfacher und problemloser gestaltet sich die Implementation. Schon jetzt zeichnen sich Entwicklungen ab, bei denen der objektorientierte Entwurf „gezeichnet“ und hernach per Mausklick in eine Programmiersprache (z.B. C++) „übersetzt“ wird. Allerdings ersetzen solche C.A.S.E. (Computer Aided Software Engineering) Tools keineswegs den erfinderischen Geist, der hinter einem gelungenen Entwurf stehen muß, sondern stellen bestenfalls eine direkte Übersetzung gezeichneter Entwürfe in die entsprechende Syntax der gewählten Computersprache dar. Durch den Sprachumfang von z.B. C++ spiegelt sich die Klarheit des Entwurfs direkt in dem betreffenden Quelltext der Software wider. Somit erhöht eine saubere Dokumentation des objektorientierten Entwurfs die Lesbarkeit des Codes erheblich. Die Einarbeitungszeit und Pflege sowohl des Entwurfes als auch des Codes profitieren gleichermaßen hiervon.

Wird ein objektorientierter Entwurf angefertigt, so kann der Entwickler bereits in dieser Phase eine Wiederverwendbarkeit der zu gestaltenden Module planen. Hierzu werden neben speziellen Schnittstellen auch solche für einen eher „universellen“ Gebrauch eingebaut. Auch hier stellt die strikte Trennung von Planungs- und Implementationsphase eine Erleichterung dar: Schnittstellen können vorgesehen, jedoch noch nicht notwendigerweise völlig implementiert werden. Die Schnittstellen werden dann bei Bedarf ausgebaut oder aktiviert („Genpool“). Wird beispielsweise Software objektorientiert entworfen, so wird der Lösungsweg zur Gestaltung eines komplexen Gegenstands nicht an maschinengegebene Abhängigkeiten ausgerichtet, sondern entspricht eher den evolutiven Problemlösungsmustern, die sich auf unserem Planeten schon seit mehr als 1.5 Milliarden Jahren bewährt haben.

4.1 Objekt

Objekte sind abgrenzbare Elemente in der Welt der aktuellen Problemstellung, die miteinander in Wechselwirkung stehen. Sie setzen sich aus zwei logischen Einheiten zusammen: Der „reaktiven“ Einheit ihrer Funktionen und der „passiven“ Einheit ihrer Daten. Theoretisch könnte es Objekte geben, die nur aus einer Ansammlung von Funktionen oder Daten bestehen. Dies dürfte jedoch die Ausnahme in sehr speziellen Problemstellungen sein. Objektorientierte Programmierung bedeutet, daß sich diese Technik zur Erstellung entsprechender Computerprogramme streng an solchen Objekten ausrichtet. Die zugehörigen Programmiersprachen unterstützen diese in ihrem Sprachumfang (CLOS [Kee89], C++ [Str92] , DSM [Sha89], Eiffel [Mey88], Smalltalk-80 [Gol83], Trellis-Owl [Sch86]). Aufgrund seiner modularen Konzeption können auch unter Ada [Cul92] objektorientierte Programme geschrieben werden, wenn auch keine besonderen Sprachkonzepte, insbesondere die Vererbung betreffend, vorliegen, wie dies bei den explizit unterstützenden Programmiersprachen der Fall ist. Dank der hervorragenden Kapselung (vgl. Kapitel 4.5 „Kapselung“) können unter Ada besonders große Programme geschrieben werden, die durch die OMT (**O**bject **M**odeling **T**echnique) [Rum91] auch objektorientiert konzeptionierbar sind. Zur graphischen Darstellung der Objekte gibt es unterschiedliche Notationsformen ([Boo94], [Boo95], [Coa90], [Rum91], [Shl88]). Die vorliegende Arbeit verwendet die Notation von [Rum91]. Danach ist das Objektorientierte Design nur dann ausreichend beschrieben, wenn drei Ansichten der aktuellen Problemstellung dokumentiert werden:

- *Objektmodell*
- *Dynamisches Modell*
- *Funktionales Modell*

Beim *Objektmodell* werden die Objekte und ihre („phylogenetischen“) Beziehungen zueinander als zentraler Punkt beschrieben. Das *Objektmodell* stellt immer wieder die „besteht aus...“ oder „wird vererbt von...“ Gesichtspunkte heraus. Er-

gebnis des *Objektmodells* ist eine statische Struktur als Lösung des beschriebenen Problems. Das *dynamische Modell* betrachtet ausschließlich den Informationsfluß zwischen den Objekten. Es beschreibt die zeitabhängigen Ereignisse, die zwischen den Objekten auftreten können, wenn das Programm läuft.
Charakteristisch für das *dynamische Modell* sind somit Ereignisse, aktuelle Stati von Objekten als Ergebnis von Interaktionen zwischen diesen. Im *funktionalen*

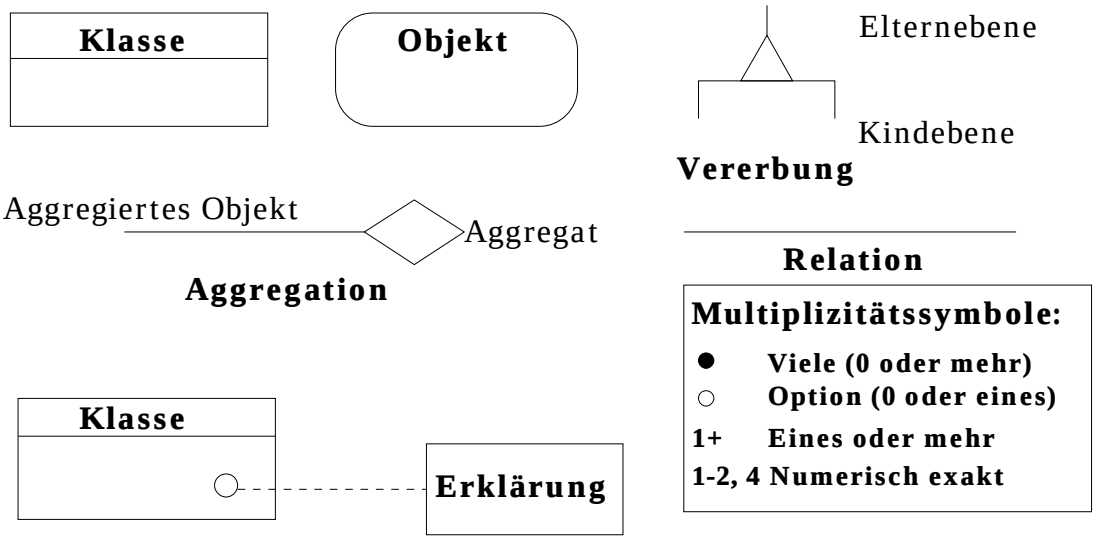


Abbildung 4-1: Graphische Notation für einige Begriffe des objektorientierten Entwurfs. Eine ausführliche Darstellung findet sich bei [Rum91].

Modell wird beschrieben, wie Ausgabewerte über bestimmte Operationen aus entsprechenden Eingabewerten erhalten werden. Nach [Rum91] werden Objekte durch ein Rechteck mit abgerundeten Ecken dargestellt. Innerhalb dieses Symbols findet sich der Objektname (vgl. Abbildung 4-1).

4.1.1 Objektgranularität

Soll ein System beschrieben werden, so gibt es unzählig viele Möglichkeiten Objekte festzulegen. Für ein gegebenes System kann man zahlreiche kleine Objekte finden, oder einige große definieren, welche die gleichen Aufgaben übernehmen. Die passende „Granularität“ eines Objektes wird ein Kompromiß sein zwischen größtmöglicher Flexibilität (was meist zu kleineren Objekten führt) und festgelegter Funktionalität (was meist größere Objekte entstehen läßt).

4.1.2 Objektschnittstelle

Jede von einem Objekt deklarierte Operation spezifiziert folgende Signatur:

- Operationsnamen

- Objekte als Parameter übergeben
- Rückgabewert

Somit definiert die Signatur über die erforderlichen Argumente (Typen, Anzahl und deren Reihenfolge) und die zurückgegebenen Werte eine Schnittstelle zur Operation. Die Schnittstelle des Objektes legt die Art der Anfragen fest, die an dieses Objekt gerichtet werden können. Schnittstellen können andere Schnittstellen als Untermengen enthalten. Objekte sind ihrer „Außenwelt“ nur über ihre Schnittstellen bekannt. Schnittstellen können auch von anderen Objekten „geerbt“ werden (vgl. 4.4 „Vererbung“). Alle Operationen oder Anfragen, die von einem Objekt an ein anderes gerichtet sind, können nur über die Schnittstelle des befragten Objektes erfolgen. Die Schnittstelle der Objekte läßt sich modellhaft mit einer selektiv permeablen Membran in der Biologie vergleichen. Die Rezeptoren sorgen dafür, daß eine genau definierte Schnittstelle zum umgebenden Milieu vorliegt. Rezeptorbindungsstellen, beispielsweise für Transmitter in der Membran, bewirken bestimmte Aktionen (Operationen) im Zellinneren, wie z.B. Erhöhung der Kinaseaktivität. Die Objektschnittstelle garantiert, daß die inneren Angelegenheiten einer Klasse von außen nicht (unkontrolliert) verändert werden können und trägt somit zu dessen Konsistenzsicherung bei. Man spricht von der „Kapselung des Objektes“ bei (vgl. Kapitel 4.5 „Kapselung“).

4.2 Klasse

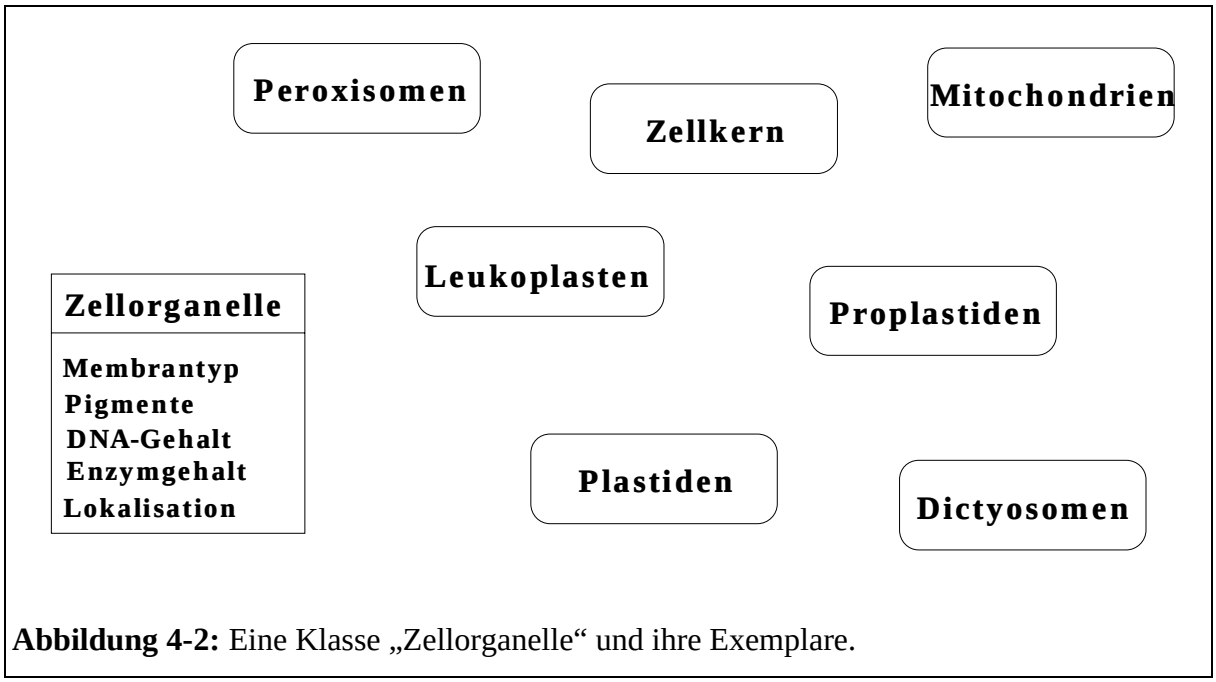


Abbildung 4-2: Eine Klasse „Zellorganelle“ und ihre Exemplare.

Lassen sich Objekte aufgrund ihrer Eigenschaften zu Gruppen zusammenfassen, so bezeichnet man die abstrakte Darstellung hiervon als Klasse. Somit ist eine Klasse eine „Schablone“, die das Verhalten (Operationen) und Attribute (Eigenschaften) ihrer zusammengefaßten Objekte abstrahiert. Die Individualität

der von ihr zusammengefaßten Objekte ergibt sich daraus, daß diese Objekte verschiedene Verhaltens*muster* und verschiedene Attribut*werte* haben (vgl. Abbildung 4-2). Aus einer Klasse leitet man die Objekte ab. Im Englischen wird dieser Sachverhalt wie folgt beschrieben: „...object is an *instance* of a class" [Rum91]. Die korrekte deutsche Übersetzung ist nicht „*Instanz*" sondern „*Exemplar*" [Gam96, S.17[1]], da „*Instanz*" nicht die konkrete Ausprägung einer Klasse zum Ausdruck bringt.

4.3 Relationen

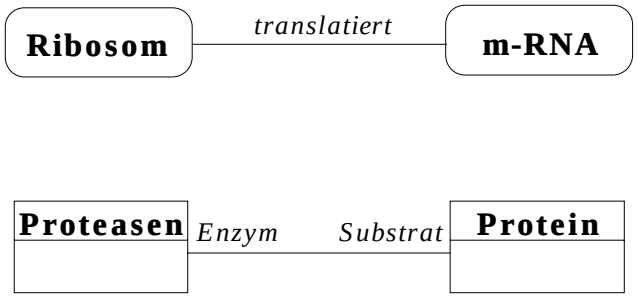


Abbildung 4-3: unidirektionale Verknüpfung (oben) und bidirektionale Assoziation (unten).

Um dem Unterschied Objekt-Klasse gerecht zu werden, bezeichnet man die Relationen zwischen Objekten als „Verknüpfung", die Relation zwischen Klassen als „Assoziation". Die „Verknüpfung" ist also ein Examplar von „Assoziation". Das Symbol hierfür ist eine einfache Linie, wobei auf der Linie Bezeichner für die jeweiligen Rollen plaziert sind. Die einfache Relation kann bidirektional oder unidirektional sein. Der oder die Bezeichner, welche auf der Linie als kurze Schlagworte vorliegen, spiegeln diesen Sachverhalt wider (vgl. Abbildung 4-3).
Bei der Aggregation besteht die Verbindung eines Objektes zu dem assoziierten Objekt genau so lange wie dessen eigene Lebenszeit, weil sie Teil von ihm ist. Bei der Assoziation kann die Verbindung schon vorher gelöst werden. Bei der Assoziationsbeziehung kennt ein Objekt das andere lediglich. Die Aggregation ist von ihrer Natur her immer unidirektional. Das Symbol der Aggregation ist eine Raute (vgl. Abbildung 4-4).

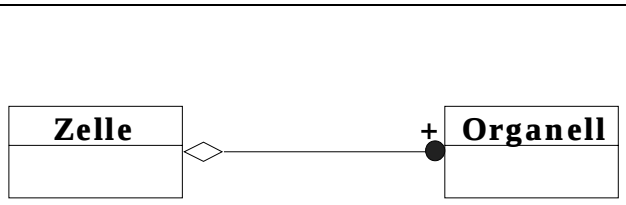


Abbildung 4-4: Aggregation. Eine Zelle besteht aus vielen Organellen.

Der schwarze Punkt mit hochgestelltem Pluszeichen in Abbildung 4-4 symbolisiert, daß es viele Exemplare der Klasse des Typs „Organell" gibt, über die ein Exemplar der Klasse „Zelle" verfügt.
Diese Multiplizitätssymbole sind in [Rum91] dokumentiert (vgl. auch Abbildung 4-1 „Multiplizitätssymbole").

4.4 Vererbung

Die Eigenschaften einer Klasse können auf eine von ihr dann als „abgeleitet“ bezeichnete Klasse übergehen. Dies hat eine phylogenetische Organisation dieser Klassen zur Folge. Man spricht von Eltern- und Kindklassen. Die Kindklasse „erbt“ das Verhalten und die Attribute der Elternklasse. Es gibt Möglichkeiten, nicht alle Elemente der Elternklasse zur freien Verwendung in der Kindklasse weiterzugeben. Diese nur den Eltern vorbehaltenen Elemente werden geschützt. In der Programmiersprache C++ steht hierfür der Bezeichner „private“ zur Verfügung [Str92]. Sollen Elemente nur den Kindern, nicht jedoch allen fremden Klassen zugänglich sein, sind diese Elemente als „protected“ deklariert. Elemente der Schnittstelle sind per definitionem „public“.

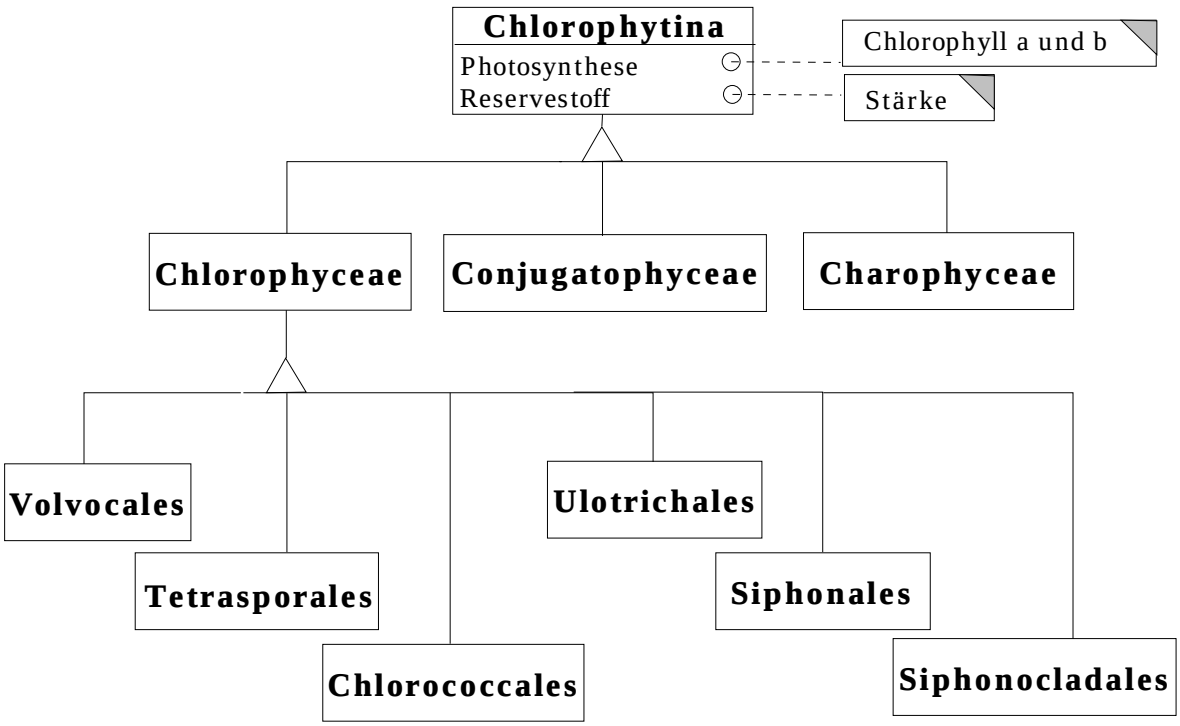


Abbildung 4-5: Vererbung anhand der Phylogenie der Grünalgen. Die geschachtelte Darstellung der Ordnungen ist rein zeichentechnisch begründet.

So ist es beispielsweise auch möglich, zur Laufzeit des Programms ein Exemplar des Typs „Bildschirmausgabe“ gegen ein Exemplar des Typs „CRS-Roboter“ auszutauschen, wenn diese von der gleichen Elternklasse („View“) abgeleitet sind. Somit werden entsprechende Ausgaben nicht mehr auf dem Bildschirm gemacht, sondern bewirken auf direktem Weg Aktionen im Roboter. Von den Elternklassen, welche in der beschriebenen Art und Weise Schnittstellen vererben, werden meist keine Exemplare erzeugt. Sie heißen dann abstrakte Klassen. Diejenigen Klassen jedoch, von denen Exemplare erzeugt werden, heißen konkrete Klassen. Das graphische Symbol für eine Vererbung ist (vgl. Abbildung 4-5) ein

aufrechtes Dreieck, das mit seiner Spitze zum Elter zeigt. Ein verbreitetes Stilmittel des OOD ist es, eine komplette Schnittstelle zu vererben. Dies hat den Vorteil, daß alle abgeleiteten Kinder die gleiche Schnittstelle haben werden. Somit können die Exemplare der Kinder über gleiche Anfragen angesprochen werden, oder sind in entsprechenden Design-Mustern gegeneinander austauschbar (z.B. „ControllerInterface“ Klasse des virtuellen Roboters, vgl. Kapitel 5.7.1.2.).

4.5 Kapselung

Interne Aspekte einer Klasse werden von allen übrigen Klassen verborgen. Die aus der Klasse abgeleiteten Objekte sind nur über ihre (öffentliche) Schnittstelle erreichbar. Somit ist sichergestellt, daß sich keine unkontrollierten Abhängigkeiten im Entwurf des Programmgefüges ergeben. Kapselung ist ein wichtiges Werkzeug, wiederverwendbare Software IC’s zu erzeugen, weil sichergestellt ist, daß Veränderungen innerhalb der gekapselten Klassen keine Seiteneffekte auf die übrigen Elemente bewirken. Somit ist die Austauschbarkeit der Einzelelemente gegen neue Versionen oder Entwicklungen, welche die gleiche Schnittstelle aufweisen, jederzeit gegeben. Eine ausführliche Diskussion über Kapselung ist bei [Mic88] beschrieben.

5 Automation einer Plasmidpräparation

5.1 Darstellung des Gesamtsystems

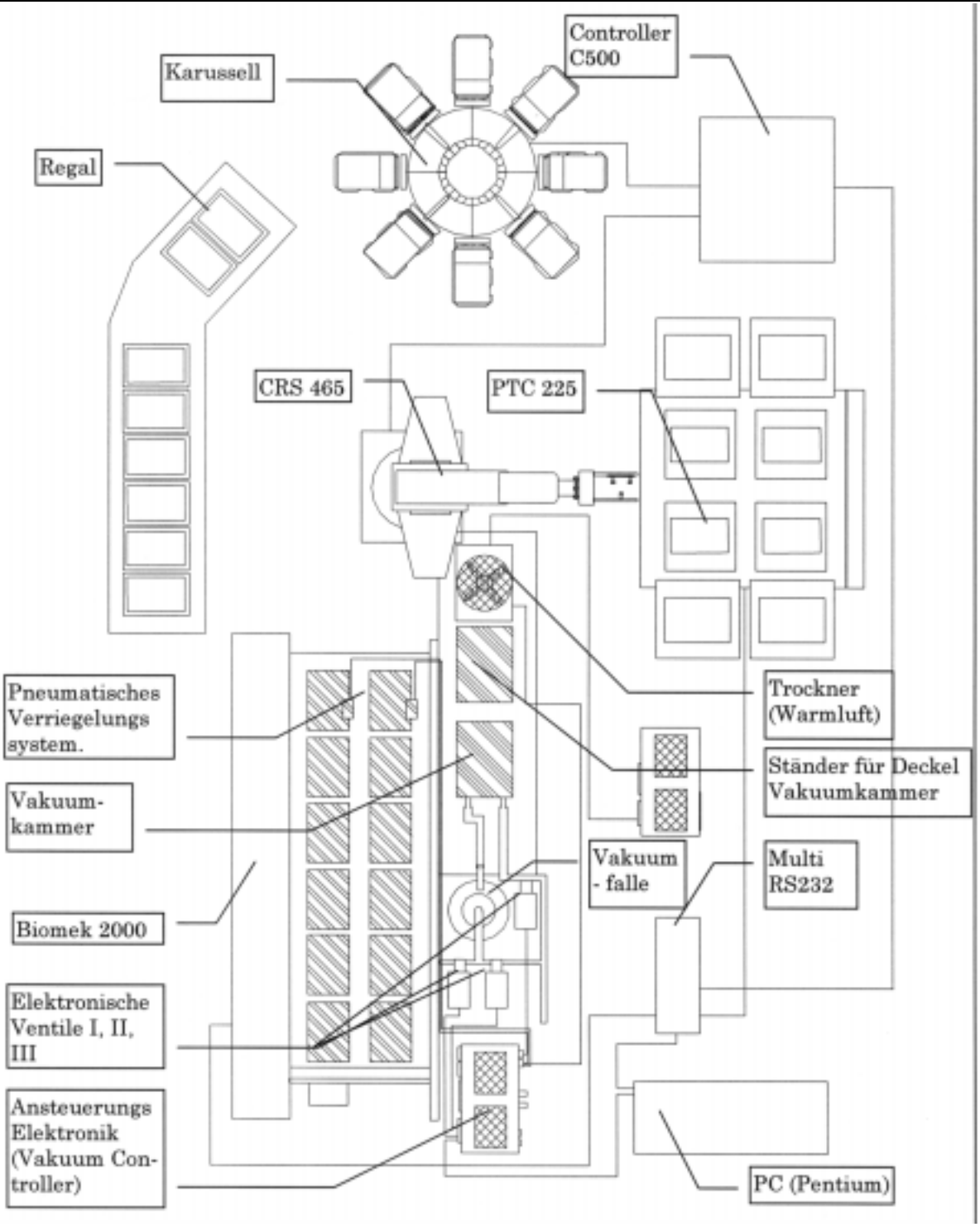


Abbildung 5-1: Darstellung der beteiligten Roboter und Einrichtungen.

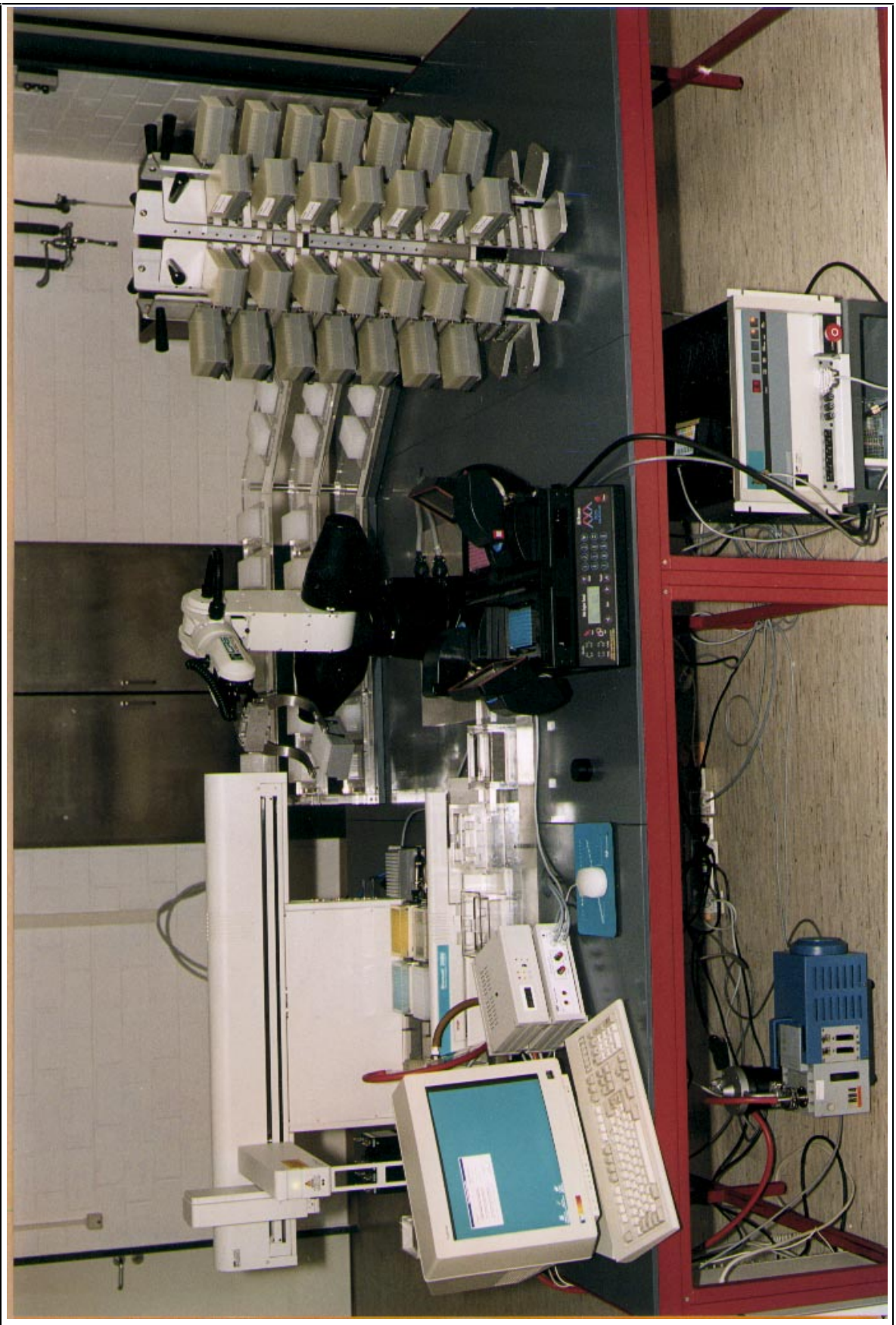


Abbildung 5-2: Abbildung der beteiligten Roboter und Einrichtungen.

Das Gesamtsystem, welches in dieser Arbeit für die automatisierte Präparation von Plasmiden aus lebenden Bakterien mit anschließender PCR-Sequenzierung verwendet wurde, ist in Abbildung 5-1 (bzw. Abbildung 5-2) dargestellt. Das Karussell führt während des Betriebs der Anlage rasche Drehbewegungen aus. Die dabei auftretenden Beschleunigungskräfte sind groß genug, um dort eingestellte Lösungen aus ihren Behältern zu vergießen. Daher werden diese im Regalbereich untergebracht, während die übrigen, zur Präparation notwendigen Materialien im Karussel lokalisiert sind. Für eine PCR-Sequenzierung aus lebenden Bakterien ist der Arbeitsablauf des Gesamtsystems wie folgt (Qiagen Turbo-Prep):

1. Manuelle Bereitstellung der zur Präparation notwendigen Chemikalien in speziellen Gefäßen. Die Hochsalzpuffer und die lebenden, sedimentierten Bakterien (ohne Überstand) werden in das Regalsystem eingestellt, während Filter, Pipettenspitzen, Mikrotiterplatten usw. in das automatische Karussell plaziert werden.
2. Startbefehl an der Steuerungssoftware (Schaltfläche PCR-Sequenzierung)
3. Der CRS 465 stellt aus dem Karusell Pipettenspitzen an die pneumatisch verriegelbaren Positionen des Biomek 2000 ein.
4. Der CRS 465 bringt das weiße Qiafilter aus dem Karussell auf die Pipettierfläche des Biomek 2000, das blaue Filter hingegen in die Vakuumkammer, Ebene 1.
5. Aus dem Regalbereich transportiert der CRS 465 die zur Präparation notwendigen Lösungen auf den Pipettierbereich des Biomek 2000.
6. Der Biomek 2000 prozessiert die Lyse der Bakterien.
7. Der Biomek 2000 transportiert das Lysat in das weiße Qiafilter.
8. Der CRS 465 stellt das weiße Qiafilter von der Pipettierfläche des Biomek 2000 auf das blaue Qiafilter, das sich schon in der Vakuumkammer befindet.
9. Der CRS 465 schließt die Vakuumkammer, indem der Deckel von seinem Wartebereich auf die Kammer gesetzt wird.
10. Die Steuerungssoftware startet den Filtrationsprozeß in der Kammer aus Ebene 0 in Ebene 1; anschließend werden die Lösungen in den Bodenbereich durchgesaugt.
11. Nach erfolgter Filtration öffnet der CRS 465 die Vakuumkammer, indem der Deckel in seine Warteposition auf dem seitlich angebrachten Ständer transportiert wird. Das weiße Filter wird vom CRS 465 aus dem System entfernt. Das blaue Filter wird auf die Pipettierfläche des Biomek 2000 eingestellt.
12. Der Pipettierroboter appliziert den ethanolhaltigen Waschpuffer gemäß Qiagenprotokoll in das blaue Filter.
13. Der CRS 465 transportiert das blaue Filter in die Vakuumkammer.
14. Die Steuerungssoftware startet den Waschvorgang der Proben durch Anlegen von Vakuum im unteren Kammerbereich. Die Waschlösung wird durch

die Filter gesaugt und über den unteren Auslaß in die Vakuumfalle transportiert.

15. Der CRS 465 entnimmt das blaue Filter und stellt es auf den Pipettierroboter zurück.
16. Der Pipettierroboter appliziert erneut den Waschpuffer gemäß Protokoll.
17. Der CRS 465 transportiert ein zweites Mal das Filter in die Kammer, um den Waschprozeß für diesen Schritt abzuschließen.
18. Die Steuerungssoftware legt hierzu Vakuum im unteren Kammerbereich an. Die Waschlösung wird durch die Filter gesaugt und über den unteren Auslaß in die Vakuumfalle transportiert.
19. Um Ethanolreste aus den Kragenbereichen der Auslaßdüsen zu entfernen wird das Filter in einen Ventilationsbereich gestellt, indem 50-55 °C warme Luft für 5 Minuten mit einer Rate von 166 m^3/h auf das Filter geblasen wird.
20. Das Filter wird erneut in die Vakuumkammer gestellt und durch Anlegen von verstärktem Unterdruck für 5 Minuten ein Luftstrom durch die Filter erzeugt. Dadurch soll Ethanol aus dem Filtermaterial entfernt werden.
21. Die Schritte 19 und 20 werden wiederholt.
22. Nach Beendigung des Trocknungsvorganges wird das blaue Filter vom CRS 465 auf den Biomek 2000 zurücktransportiert.
23. Der CRS 465 stellt den Eluatfänger (modifizierte Mikrotiterplatte) in die Vakuumkammer ein.
24. Der Biomek 2000 appliziert Elutionslösung (H_2O bidest.) in das blaue Filter.
25. Der CRS 465 transportiert den Filterträger auf den Eluatfänger und schließt in beschriebener Weise die Kammer.
26. Die Steuerungssoftware startet den Transfer des Eluates in den Eluatfänger.
27. Der CRS 465 öffnet die Kammer und entfernt das blaue Filter aus dem System.
28. Der Eluatfänger wird auf den Biomek 2000 gestellt.
29. Der CRS 465 entfernt nicht mehr benötigte Labware aus dem System.
30. Der CRS 465 stellt aus dem Regalbereich und dem Karussell für die PCR-Sequenzierreaktion notwendiges Zubehör ein.
31. Der Biomek 2000 pipettiert das PCR-Sequenzier-Reaktionsprotokoll.
32. Der CRS 465 stellt die PCR-Mikrotiterplatte in den PTC 225 ein.
33. Die Steuerungssoftware schließt die beheizbaren Deckel der PTC 225 und startet das Cycle-Sequencing.

5.2 Konstruktion einer Vakuumkammer mit zwei unabhängig von einander einstellbaren Vakuumbereichen

In dieser Arbeit werden die Qiagen-Protokolle zur Präparation von Plasmiden eingesetzt. Für die Automatisierung sind einige Modifikationen vorgenommen worden (vgl.Kapitel 5.3). Um einen möglichst hohen Probendurchsatz zu erreichen, wurden die Filterträger des Qiawell 96 Turbo Plasmid Kits eingesetzt, die ein 96er Raster im Mikrotiterplattenformat aufweisen.

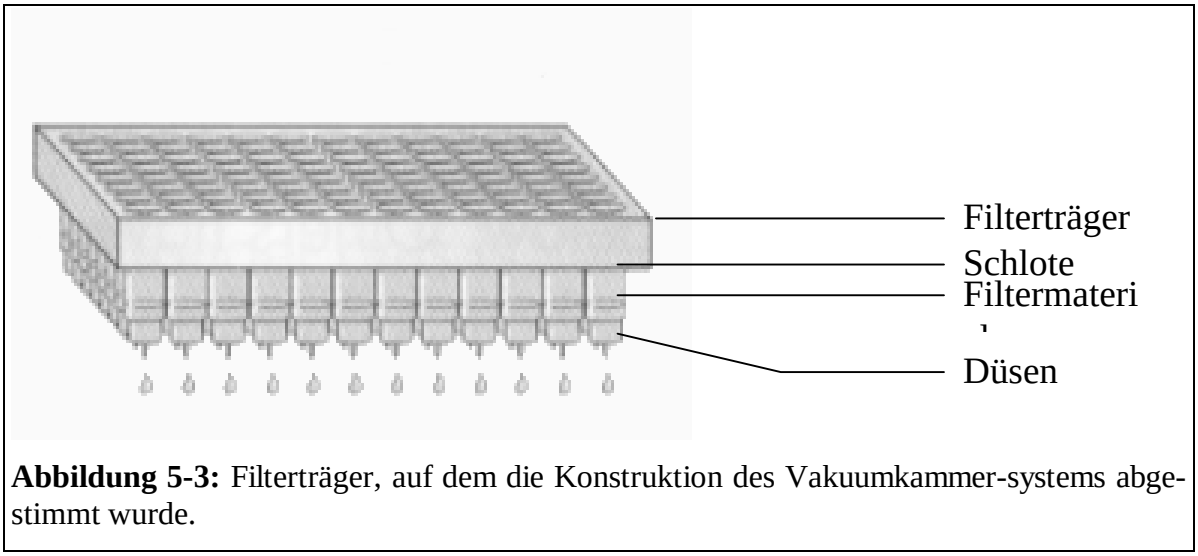


Abbildung 5-3: Filterträger, auf dem die Konstruktion des Vakuumkammer-systems abgestimmt wurde.

5.2.1 Wahl des Fertigungsmaterials für die Vakuumkammer

Die Wahl des Fertigungsmaterials wurde in der Konstruktionsphase von folgenden Gesichtspunkten geleitet: Zum einen sollte das Material gegen alle Chemikalien, die während des Präparationsprozesses die Oberflächen der Kammer berühren, resistent sein. Zum anderen sollten sich konstruktionsbedingte Verbesserungen rasch und einfach verwirklichen lassen. Das Basismaterial sollte steif sein, damit unter dem Einfluß des sich in der Kammer aufbauenden Vakuums keine Verformungen ergeben können. Somit schieden viele Kunststoffe als Material für das Labormuster aus. Für eine spätere industrielle Serienproduktion standen ökonomische Gründe gegen die Verwendung von Edelstahl. Für die industrielle Produktion sind Gießverfahren interessant, weil sie die aufwendigen und zeitintensiven Fräsmethoden ersetzen. Um diesem Aspekt zu verwirklichen war es das Ziel des Designs, auf bewegliche Teile völlig zu verzichten. Die Transparenz sprach letztendlich auch für die Verwendung von „Plexiglas“ als Grundstoff für die Herstellung des Labormustes. So ist es sehr hilfreich, wenn die aktuelle Position des Greifroboters innerhalb der Vakuumkammer während der „Lernphase“ des Systems optisch kontrolliert werden kann. Das Labormuster wurde in der hauseigenen Werkstatt aus einem 13 x 10.5 x 16 cm großen Plexiglasblock über eine numerische Fräsanlage hergestellt. Die Konstruktionspläne wurden nach der Fertigstellung des Labormusters mit AutoCAD®, Version 13 gezeichnet [Fer97]. Nach

diesen Bauplänen wurden schließlich von einem kunststoffverarbei-tenden Betrieb drei Kammern gebaut, die in den nachfolgenden Routinepräparationen eingesetzt wurden.

5.2.2 Plazierung der Vakuumkammer auf dem Robotertisch

Die zu konstruierende Vakuumkammer sollte Teil eines größeren Robotersystems sein, das optional zur Unterstützung aller gängigen Präparationsmethoden molekularbiologischer Verfahren eingesetzt werden soll. Deswegen wurde ein modulares Design verwirklicht, das es ermöglicht, seine apparativen Komponenten auf die jeweilige Fragestellung einzurichten. Das eingesetzte Robotersystem ist nicht rückgekoppelt. Das bedeutet, daß keine visuelle oder andere sensorische Kontrolle über den aktuellen Ist-Zustand möglich ist. Alle beweglichen Komponenten dieses Systems müssen sich daher an möglichst exakt definierten Positionen befinden. Werden Komponenten durch einen Roboterarm bewegt, dann ist es von entscheidender Bedeutung, daß sich diese bewegten Komponenten bei ihrer erneuten Wiederaufnahme durch den Roboter an einer genau definierten Position befinden. Die Vakuumkammer sollte zudem als Modul aus dem System entfernbar sein, damit an ihrer Position andere Modulsysteme, für weitere Methoden eingesetzt werden können. Die Plazierung aller Hilfssysteme für die automatisierte Präparation orientiert sich am Pipettierroboter. Alle Koordinatensysteme[2] sind relativ zu diesem ausgerichtet. An seiner Gundplatte wurden mechanische Einhängepunkte angebracht, die mit hoher Präzision gefertigt wurden. Die Toleranz dieser Bezugspunkte ist besser als 1/100 mm. Ein Plattensystem aus 3 cm dicken, präzisionsgefrästem Plexiglas ermöglicht die ausreichend genaue Positionierung von Modulen. Die Bohrungen zur Aufnahme der Führungshülsen, welche es gestatten die Vakuumkammer reversibel in das System einzustellen sind so orientiert, daß der für die Beschickung des Pipettierroboters verantwortliche Greifroboter alle notwendigen Manipulationen an der Vakuumkammer vollziehen kann. Weil alle mechanischen Aktionen, die vom Greifroboter innerhalb der Vakuumkammer ausgeführt werden, nur geringfügige Toleranzen von weniger als 1/10 mm erlauben, wurde die Vakuumkammer möglichst nahe am Einhängungspunkt des Plattensystems am Pipettierroboter gewählt. Weil das Plattensystem geometrisch einem langen Hebel ähnelt, ist dort die relative Abweichung am geringsten und daher die relative mechanische Ortsposition am exaktesten, was die Reproduzierbarkeit der reversiblen Einhängung in das System anbelangt.

5.2.3 Das Design der Vakuumkammer muß auf unüberwachte Automation abgestimmt sein

Prinzipiell wäre es möglich, anstelle eines Vakuums eine Überdruckvorrichtung zu schaffen, welche die Flüssigkeiten aus den Schloten des oberen Filterhalters in die darunterliegende Region pumpen würden. Ventile wären dann für die Entlüf-

[2] Der CRS-Roboter alleine verfügt über 4 verschiedene Koordinatensysteme (vgl. Einleitung).

tung der betreffenden Bereiche zuständig. Nachteil dieser Überdruckmethode ist, daß die Luft aus einem Pumpensystem stets mit kleinsten Partikeln unbekannter Chemie verunreinigt ist. Aufwendige Filtersysteme, die einem entsprechenden Wartungsplan unterliegen, müssen im System installiert werden. Ein „Überdruckstempel“, der auf dem Filterträger aufliegt müßte mit entsprechender Kraft einwirken. Somit wäre entweder ein Hebesystem für diesen Stempel notwendig, oder der Greifroboter wäre für die gesamte Zeit der Filterung blockiert. Ersteres ist mechanisch aufwendig, teuer in der Herstellung und durch seine beweglichen mechanischen Teile wartungsbedürftig. Letzteres ist unökonomisch, da der Greifroboter in der Zeit der Filtration andere Aufgaben, wie z.B. die Beschikkung des Pipettierroboters oder der PRC-Maschine übernehmen kann, was die Betriebskosten der Gesamtanlage vermindert. Diese Gründe sprachen für die Entwicklung eines Unterdrucksystems. Es hat das Ziel, Flüssigkeit in zwei zeitlich wohldefinierten Schritten aus einem Filter in einen darunter befindlichen zu transportieren, um von dort in den untersten Bereich der Kammer abgesaugt zu werden. Um die zwei Vakuumbereiche, in denen sich die Filterhalter befinden, unabhängig voneinander einzustellen ist es notwendig, die Kammer in zwei Bereiche zu unterteilen, die den jeweiligen Filterträger aufnehmen und gegen den übrigen Bereich abdichten. Die Zone zwischen den Filterträgern bzw. zwischen dem Filterträger und dem Boden der Kammer sind mit Anschlüssen zur Evakuierung der hier befindlichen Atmosphäre versehen. Prinzipiell ist es möglich, den obersten Filterhalter auf einen Deckel sitzend einzustellen, in dem der Deckel als Filterwanne mit einer entsprechend vertieften Dichtung dient. Ein hängendes System gewährleistet jedoch nicht den ausreichenden Anpreßdruck für die Abdichtung des Filterhalters gegen das übrige System der Vakuumkammer während der Einstellung des Vakuums. Um einen Filterträger mit einem Roboter in die Vakuumkammer einzustellen, bedarf es geeigneter mechanischer Führungen, damit die Toleranzen ausgeglichen werden, die beim Führen oder Greifen des Trägers stets auftreten können. Bei einem aufsitzenden Deckel ist das Aufbringen des Filters über einen Roboter ohne weitere Führungszusätze nicht sicher. Selbst wenn man äußere Phasen anbrächte, was fertigungstechnisch teuerer wäre als bei einem innenzentrierenden Deckel, ist die automatische Stabilisierung der inneren Komponenten durch einen umschließenden Deckels vorzuziehen. Bei einem aufsitzenden Filterhalter wäre es notwendig, daß der Greifroboter den Filterhalter immer dann an den oberen Deckel preßt, wenn das System belüftet wird. Weil sich während der ersten 5 bis 10 Sekunden über die noch unbenetzten Schlöte des unteren Filterhalters ein Vakuum im Bodenbereich einschleicht (s.u. 5.2.3.5, Seite 47), ist eine zyklische Entlüftung als Sicherheitsmaßnahme notwendig. Weil letzteres als Schutz gegen zu erwartende Fertigungstoleranzen und Ermüdungserscheinungen von Dichtungsmaterialien im Ansteuerungsprotokoll vorgesehen ist, wäre der Greifroboter während der Filtrationszeit ständig mit diesen Arbeiten beschäftigt, was ökonomisch nicht sinnvoll ist. Als Konsequenz wurde ein umschließender Deckel konstruiert, der über sein eigenes Gewicht einen geeigneten Anpreßdruck

für die Abdichtung des Filterhalters bereitstellt. Ein umschließender Deckel weist in Verbindung mit dem Bodenteil der Vakuumkammer eine innere mechanische Stabilisierung seiner Komponenten auf. Neben der Abdichtung des Vakuums gegen den äußeren Luftdruck kommt der Deckelkomponente die wichtige Aufgabe zu, einen eventuell fehleingestellten oberen Filterträger in seiner Position noch zu korrigieren. (s.u. 5.2.3.1, Seite 35)

5.2.3.1 Passive mechanische Korrekturen während des Einbringens von Filterträgern durch einen Roboterarm

Werden die Filterhalter aus dem Pipettierroboter in die Vakuumkammer eingestellt, dann können sich auf dem Verfahrweg geringfügige Lageänderungen ergeben. Würden diese Veränderungen nicht behoben, dann käme es in einem sensorisch nicht rückgekoppelten Robotersystem, wie es in diesem Fall vorliegt, rasch zur mechanischen Katastrophe. Somit mußten an jedem Ablagepunkt passive mechanische Korrekturen konstruiert werden, die auf einfache Weise Fehlpositionierungen des transportierten Objektes ausgleichen. Zu diesem Zweck wurden für die Arbeitsfläche des Pipettierroboters spezielle Einstellschalen entwickelt, welche die Filterträger am Pipettierroboter aufnehmen. Zusammen mit speziell konstruierten Hülsen am Filterträger sorgen diese für eine exakte Positionierung (s.u. 5.2.3.1.1, Seite 42). Die Einstellschalen werden ebenfalls aus seitlichen Regalbereichen über den Greifroboter auf den Pipettierroboter gebracht. Für deren eigene Zentrierung sorgen sowohl die gefederten Aufnahmen am Pipettierroboter als auch kurze 30° Phasen an den Längskanten der Schalen selbst. Durch diese Vorrichtungen ist eine ausreichend präzise und repetitive Positionierung auf dem Pipettierroboter gewährleistet. Je geringer die Varianzen bei der Aufnahme der Filter hier sind, desto einfacher lassen sich die passiven Führungselemente an der Vakuumkammer gestalten. Wenn der Greifer des CRS-Roboters den Filterträger auf den Pipettierroboter einstellt, so erfolgt der erste mechanische Kontakt über die 45° Phasen an den Schächten der Einstellschale (Abbildung 5-4: Teil 1) und den korrespondierenden 45° Phasen der Abstandshülsen, die sich am Filterträger in den vier Eckpositionen befinden. Durch die Abwärtsbewegung des Roboterarms werden bis zu ca. 3mm fehlplazierte Filterhalter noch in die richtige Endposition gebracht. Die erforderlichen Korrekturkräfte ergeben sich durch die breit anliegenden Seitenwände der Abstandshülsen einerseits und die entsprechenden Aussparungen an der Filterschale andererseits. Die Plasmidlösungen werden im letzten Schritt der Präparation in PCR- Reaktionsgefäßen aufgefangen, für die ein spezieller Träger konstruiert wurde, an dem sich allerdings keine Abstandshülsen befinden. Seine korrekte Endposition wird über die Phasen an der Seite der Einstellschale erreicht (Abbildung 5-4: Teil 2).

Wird das die Flüssigkeit auffangende Filter in die Vakuumkammer eingestellt, so sind es die Führungslaschen (Abbildung 5-5, Teil 1; Abbildung 5-6), welche den ersten mechanischen Kontakt zur Vakuumkammer herstellen. Weil die Filterhal-

ter während des Verfahrweges von der Pipettierstation zur Vakuumkammer geringfügige Lageänderungen erfahren können dienen diese Führungslaschen dazu, den Träger während des Einführens in den Unterteil der Kammer in die erforderliche Lage zurückzubringen. Der in dieser Arbeit verwendete CRS 465 kann vorgegebene Koordinaten nur über eine im C500 Controller berechnete „Spline-Funktion“ anfahren. Deswegen ist es notwendig, stets hinsichtlich der „Spline-Funktion“ entspannte Bewegungsabläufe vorzugeben. Somit nähert sich der Robotergreifarm mit dem Filterhalter zunächst mit mittelhoher Geschwindigkeit einer Annäherungsposition ca. 1 cm senkrecht über dem endgültigen Kontaktpunkt zu den Führungslaschen. Beim langsamen Absenken der Filterträger wird der Kontakt an den Phasen der Laschen erfolgen, wobei Fehlstellungen zunächst an den 30° Phasen, später an den 2° Neigungen der senkrecht zur Bewegungsrichtung ausgerichteten Flächen über die erzwungene Abwärtsbewegung des Greifarms ausgeglichen werden.

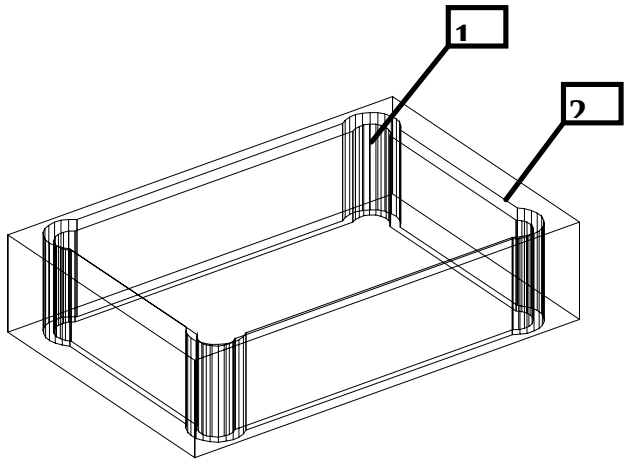


Abbildung 5-4: Einstellschale zur Aufnahme eines Filterhalters mit aufgesetzten Abstandshülsen.

1= Schacht zur Aufnahme der Abstandshülse am Filterhalter.
2= Phase zur Zentrierung fehlplazierter Produktträger.

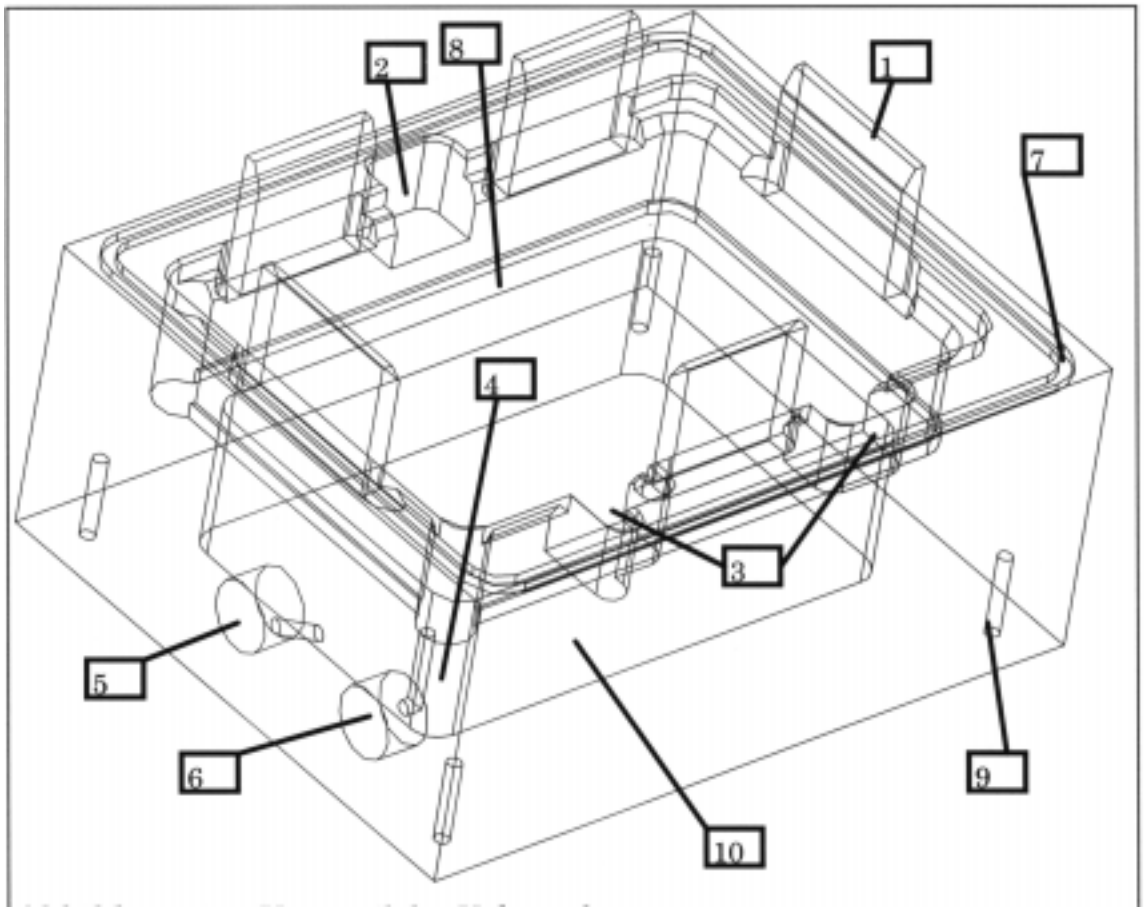


Abbildung 5-5: Unterteil der Vakuumkammer.

1= Führungslasche für die Filterhalter mit korrigierenden Phasen.
2= Aussparung für den CRS Greifer links.
3= Aussparung für den CRS Greifer rechts.
4= Zentrierungsschacht für den Abstandshalter des Filters.
5= Absaugschacht unterer Filterbereich, dient auch der Abfallentsorgung.
6= Absaugschacht oberer Filterbereich.
7= Nut zur Aufnahme des Ringgummis (60 Shore).
8= Auflagefläche des mittleren Verdichtungsgummis (60 Shore).
9= Bolzen zur Fixierung der Vakuumkammer im System.
10= Kammerboden zur Aufnahme des unteren Filters.

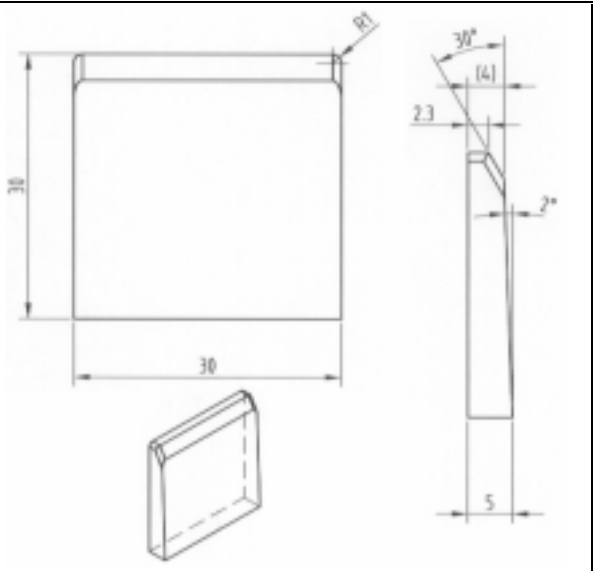


Abbildung 5-6: Führungslasche der unteren Vakuumkammerhälfte. Sie dient zur ersten Zentrierung der Filterträger während diese vom Greifarm in die Kammer eingestellt werden. Größere Lageabweichungen der Filterträger können durch sie ausgeglichen werden.

Nach ca. 1 cm ist der Filterträger von den fünf, im 2° Winkel sich nach unten verjüngenden Justierflächen (vgl. Abbildung 5-6) derart flankiert, daß Fehlstellungen im horizontalen Bereich relativ zur Vakuumkammer in jedem Fall ausgeglichen sind. Nun erst erfolgt die vertikale Justierung der Filterträger an den Zentrierschächten der Kammer. An den vier äußeren Eckschlöten des Filterträgers sind Abstandshülsen angebracht, die neben ihrer Funktion den oberen Filterträger für das Vakuum im oberen Bereich der Kammer zugänglich zu machen, hier ihre zweite Funktion wahrnehmen: Die vertikale Korrektur von Fehlstellungen des unteren Filterträgers. Hierzu befinden sich ebenfalls 30° Phasen sowohl an den Abstandshülsen als auch an den Zentrierungsschächten (Abbildung 5-5, Teil 4) des Vakuumkammer-Unterteils. Nach ca. 2 mm Verfahrstrecke ist der Filterhalter durch die Abwärtsbewegung des Roboterarmes in korrekter Lage innerhalb der Vakuumkammer positioniert. Das Greifwerkzeug fährt dabei in die Aussparungen (Abbildung 5-5, Teile 2 und 3) der Kammer ein. Etwa 1mm oberhalb des endgültigen Aufsetzpunktes wird der Filterträger vom Greifer des Roboterarmes gelöst und setzt somit auf die Abdichtung (Abbildung 5-5, Teil 8) des Kammerbodens auf. Wird der obere Filterträger über den Robotergreifarm eingestellt, so wird dessen Zentrierung in ähnlicher Weise über die Führungslaschen (Abbildung 5-5, Teil 1) bewerkstelligt. Die Abstandshülsen richten den oberen Filterträger parallel zum bereits in der Kammer befindlichen unteren Träger aus. Falls es dennoch zu einer Fehlstellung des oberen Filterhalters kommen sollte, wird diese durch die besondere Konstruktion des Vakuumkammer-Oberteils ausgeglichen. Wird das Oberteil der Vakuumkammer aus einer speziellen Parkposition vom Greifer des CRS 465 auf das Vakuumkammer-Unterteil aufgesetzt, so erfolgt der erste Kontakt mit dem obersten Filterträger an seiner Innenkante. Hier wird eine wichtige Funktion des Oberteils ausgeführt: Die Korrektur eines eventuell fehlplazierten oberen Filterhalters. Zu diesem Zweck ist dem Deckel an den entsprechenden Berührungszonen eine 30° Phase angebracht worden.

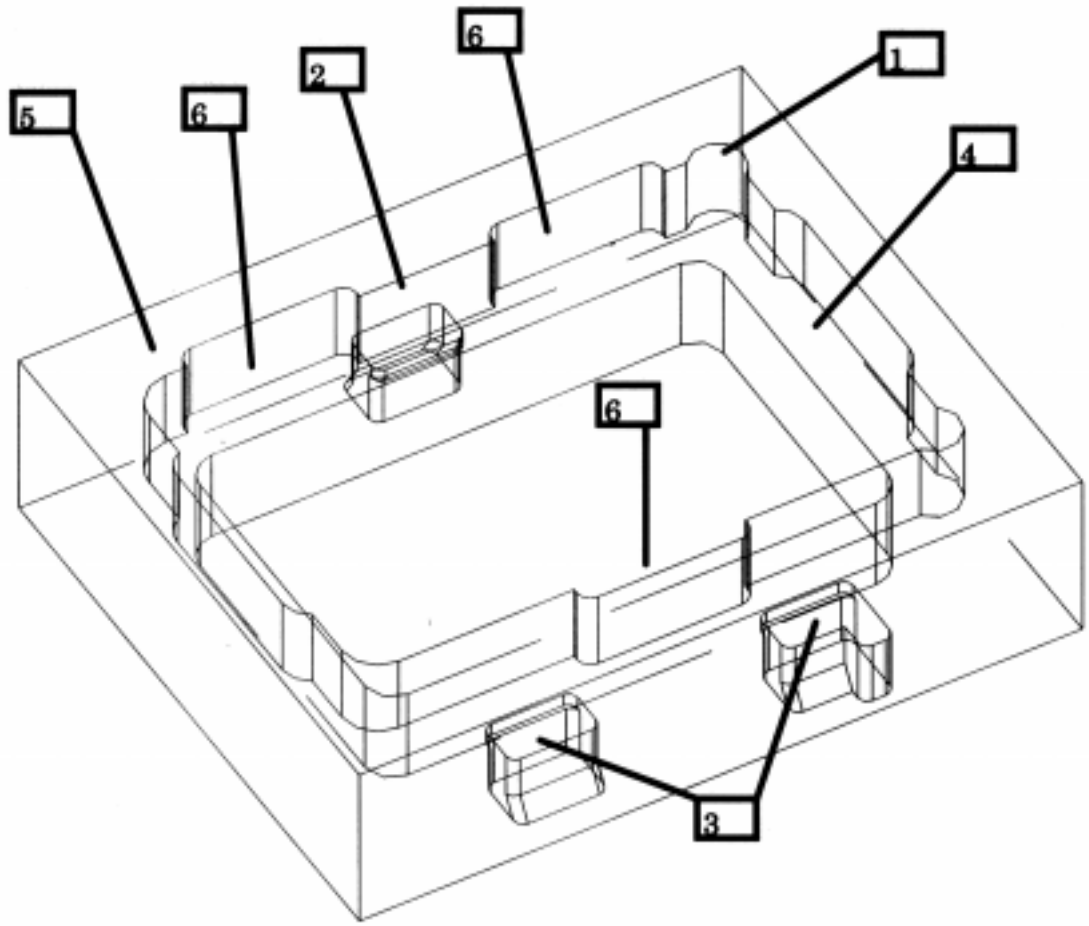


Abbildung 5-7: Oberteil der Vakuumkammer (von unten gesehen).

1= 30° Phase an der Kante zur Aufnahme des oberen Filterträgers (nicht eingezeichnet).
2= Aussparung für den CRS-Greifer links
3= Aussparung für den CRS-Greifer rechts.
4= Auflagefläche für den Filterträger. Verdichtungsmaterial 20 Shore.
5= Auflagefläche für die Verdichtung zum Unterteil. Verdichtungsmaterial 20 Shore (Gegenstück zum O-Ring 60 Shore)
6= Aufnahme für die Führungslaschen. Diese dienen zur Zentrierung des Deckels während des Aufsetzens (bevor der Kontakt zum innen stehenden Filterträger erfolgt).

Die Breite der Phase ergibt sich aus der möglichen Fehlplazierung des Filterhalters. Weil die Toleranzen über die Führungslaschen maximal 0.5mm sein können, ist die Phase am Deckel auf jeder Seite ca. 0.5 mm breit. Dies gewährleistet ausreichende Toleranz in der Aufnahme des Filterhalters. Durch die Abwärtsbewegung des Greifers wird der Filterhalter, falls notwendig, über die Phasen des Deckels in die geeignete Positon gezwungen. Der nächste Kontakt des Deckels erfolgt während seiner Abwärtsbewegung an den Führungslaschen des Unterteils. Diese haben für diesen Zweck neben ihren 30° Innenphasen noch kurze 45° Au-

ßenphasen erhalten. Der Deckel nimmt diese in speziellen Einfräsungen auf (Abbildung 5-7, Teil 6). Über die Abwärtsbewegung des Greifers wird der Dekkel an den Einsparungen ausgerichtet. Nach ca. 0.5 cm ist der Deckel mechanisch schon so weit an dem Unterteil der Vakuumkammer ausgerichtet, daß ein sicheres Aufsetzen gewährleistet ist. Die Verdichtung der zusammengesetzen Kammer erfolgt an insgesamt drei Stellen. Die unterste Zone verdichtet über die 60 Shore harte Gummiauflage. Die zweite Zone ist die „Nahtstelle“ zwischen dem Unterteil und dem Deckel der Vakuumkammer. Sie wird aus der Kombination O-Ring (60 Shore) und elastischem (30 Shore) Dichtungsband gebildet. Die dritte Zone schließt das obere Filter gegen den inneren Deckelbereich ab. Der Deckel verfügt also über zwei Auflageflächen, die für die Anbringung des Dichtungsmaterials vorgesehen sind (Abbildung 5-5, Teil 4 und 5). Sie werden zu diesem Zweck fertigungstechnisch angerauht. Der Deckel der Kammer wird über den Greifer des CRS 465 bewegt. Weil kommerziell kein geeigneter Greifer zur Verfügung stand, wurde in der vorliegenden Arbeit ein geeignetes Gerät konstruiert.
Es wurde eine dreifingrige „Hand“ verwendet, welche von ihren Dimensionen der des Beckman-Sideloaders nachempfunden wurde, damit der CRS-465 problemlos vorgegebene Labware transportieren kann. Das Greifwerkzeug des CRS 465 besteht aus sich horizontal bewegenden, motorisch angetriebenen Achsen (vgl. Abbildung 3.12). Die somit zur Verfügung stehenden Kräfte werden über eine Schwalbenschwanzführung an die dreifingrige „Hand“ weitergegeben. Zur Aufnahme des Deckels sind in diesen (Abbildung 5-5, Teil 2 und 3) entsprechende Aussparungen angebracht. Oben sind diese aus Sicherheitsgründen mit breiten Phasen versehen, damit eine eventuell fehlplazierte Greiferstellung noch auskorrigiert wird. Am Boden der jeweiligen Aussparung sind zum Kammerinneren hin 1mm tiefe und 2mm hohe Einfräsungen angebracht. Grund hierfür ist, daß der Deckel für einen ungesicherten Transport zu schwer ist. Die Finger der „Greiferhand“ haben nach innen gerichtete Vorsprünge, die genau in diese Einfräsungen passen. Schließt sich der Greifer im Deckel, so liegen diese Vorsprünge in ihren Einfräsungen, was ein Abgleiten des Deckels durch seine glatte Oberfläche ausschließt.
Wird der Deckel von der Vakuumkammer genommen, um diese zu Be- oder Entladen, so muß dieser auf einer speziellen Warteposition abgestellt werden. Damit die erneute Aufnahme reproduzierbar ist, befindet sich an der Warteposition ein speziell konstruierter Ständer. Dieser verfügt über sechs Führungslaschen mit 45° Phasen, die eine Zentrierung des Deckels während seiner Ablage garantieren.

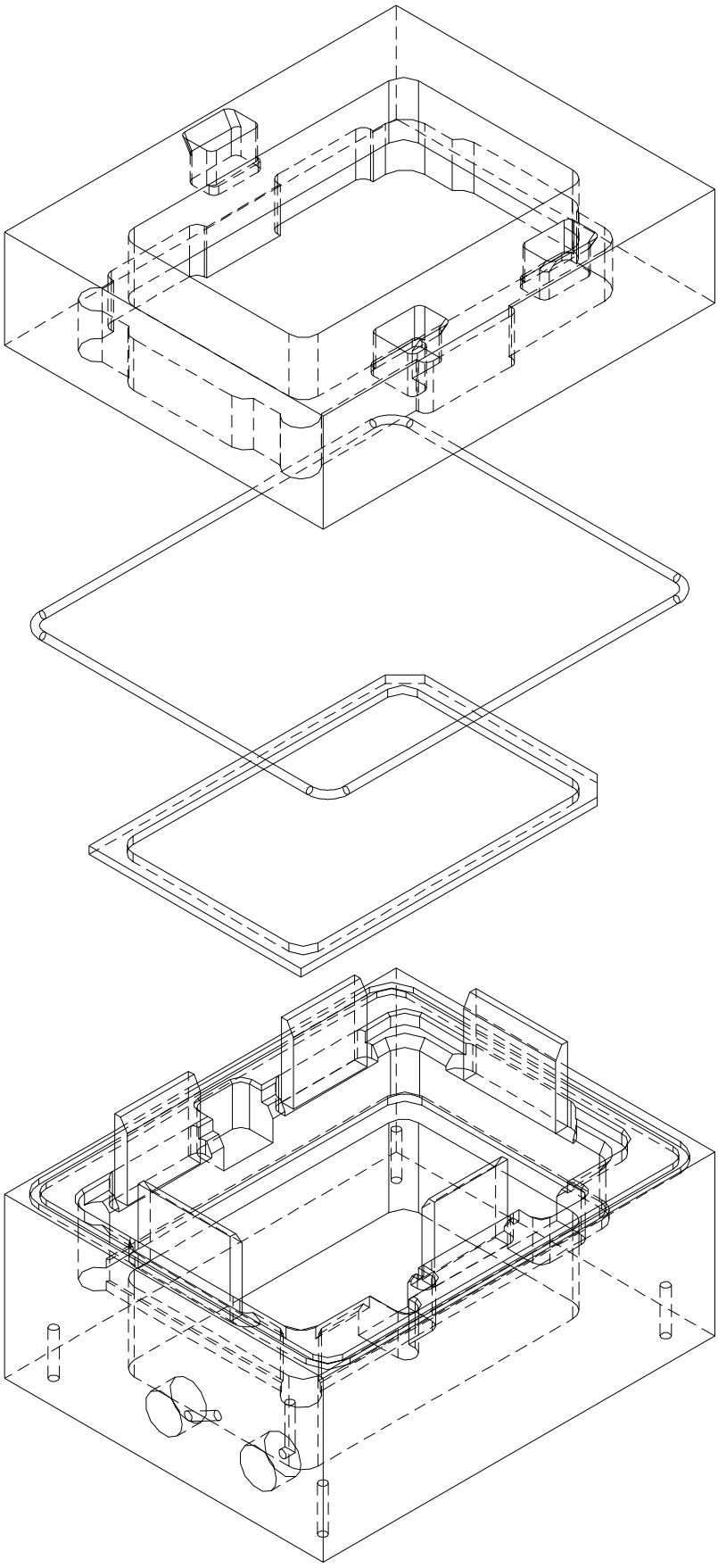


Abbildung 5-8: Explosionszeichnung der Vakuumkammer. Die Dichtungsringe im Deckel sind aus Grunden der Übersichtlichkeit nicht eingezeichnet.

5.2.3.1.1 Abstandshülsen für die Filterträger

Würden die Filterträger in der Vakuumkammer auf einfache Weise übereinander gestellt, so wäre es unmöglich, zwei getrennte Vakuumbereiche in der Kammer einzustellen. Die Auslaßstutzen der Filterträger passen dicht in die Filterschlöte des darunterbefindlichen Filters, so daß ein sich im Zwischenbereich einstellendes Vakuum nicht in der Lage wäre, die sich in den Schlöten des oberen Filterhalters befindliche Präparationsflüssigkeit abzusaugen. Deshalb ist es notwendig, einen ausreichend großen Spalt zwischen die beiden Filterträger zu legen. Andererseits ist es aus präparationstechnischen Gründen unbedingt notwendig, ein zufälliges Verspritzen der abtropfenden Flüssigkeiten in benachbarte Schlöte des unteren Filtersystems zu unterbinden. Eine Kreuzkontamination von benachbarten Proben wäre sonst die Folge, was die Ergebnisse der Präparation unbrauchbar machte. Der Abstand der beiden Filterträger wurde so bemessen, daß sich die Auslaßspitzen des oberen Filterträgers 1.5 mm innerhalb der Schlote des unteren Filterträgers befinden. Prinzipiell könnten vier Ringe, die über den verjüngten Teil der Schlote gezogen werden einen ausreichend sicheren Abstand ermöglichen. Allerdings wären diese vier Eckschlote dann nicht mehr zur Präparation geeignet, weil diese dann gegen den unteren Filterhalter abgedichtet wären. Versuche zeigten, daß auch Kanäle, welche in die Ringe gefräst wurden, nicht ausreichen, um das Vakuum ungehindert in den jeweiligen Eckschlot vordringen zu lassen. Weiterhin besteht über die besondere Geometrie der Filterhalter das Problem, daß sich dieser nur sehr schlecht über passive mechanische Konstruktionen zentrieren läßt. Die Filterhalter weisen an ihrem oberen Trägerabschnitt einen sich nach unten verbreiternden, trapezförmigen Querschnitt auf. Das Material ist hier mit 0.7 mm zu dünn, um eine ausreichende Steifigkeit für entsprechende Führungen zu bieten. Versuche zeigten, daß diese Bereiche sehr leicht brechen, wenn der Greifroboter auf der Arbeitsfläche des Pipettierroboters Fehlpositionierungen über passive Führungen ausgleichen sollte, die an diesen dünnen Wänden angriffen. Eine exakte Positionierung des Filterhalters über Einstellschalen auf dem Pipettierroboter ist jedoch unbedingt erforderlich (s.o. 5.2.3.1 Seite: 35). Die gegebenen Filterträger bieten in ihrem

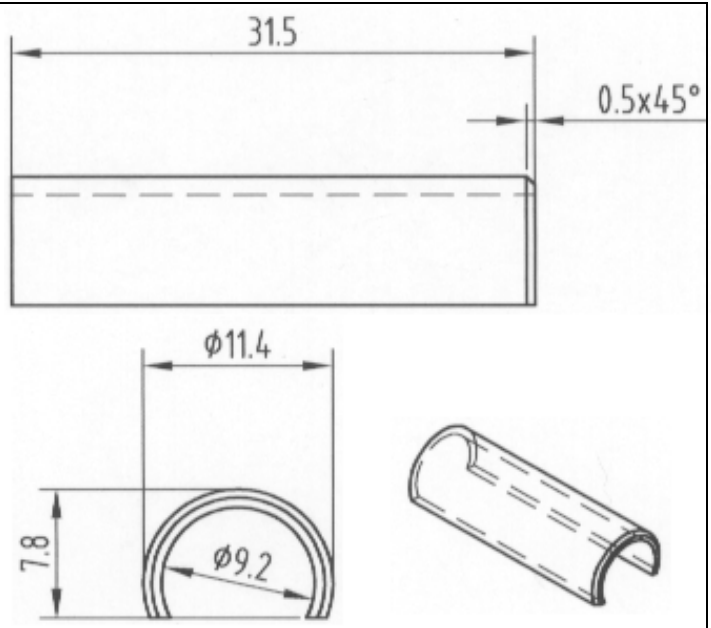


Abbildung 5-9: Abstandshülse für die Filterhalter.

unteren, kompakten Schlotbereich ausreichende Verwindungssteife für eine entsprechende Greifroboter-geführte Zentrierung. Somit wurden spezielle Abstandshülsen konstruiert, die sowohl den korrekten Abstand zwischen zwei in der Vakuumkammer übereinandergestellte Filterhalter gewährleisten, als auch Fehleinstellungen korrigieren. Letzteres wird über deren Geometrie und einen 45° Phasenanschnitt an deren unteren Rand erreicht (vgl. Abbildung 5-9). Der 45° Phasenanschnitt stellt den ersten mechanischen Kontakt zu den entsprechenden Führungen der unteren Vakuumkammerhälfte bzw. den speziellen Einstellschalen (Vgl. Abbildung 5-4) an den Parkpositionen auf der Arbeitsfläche des Pipettierroboters her. Über die Geometrie der Abstandshülsen bewirkt die weitere Abwärtsbewegung des Greifarms die notwendigen Korrekturen zur exakten Aufnahme des Filterträgers.

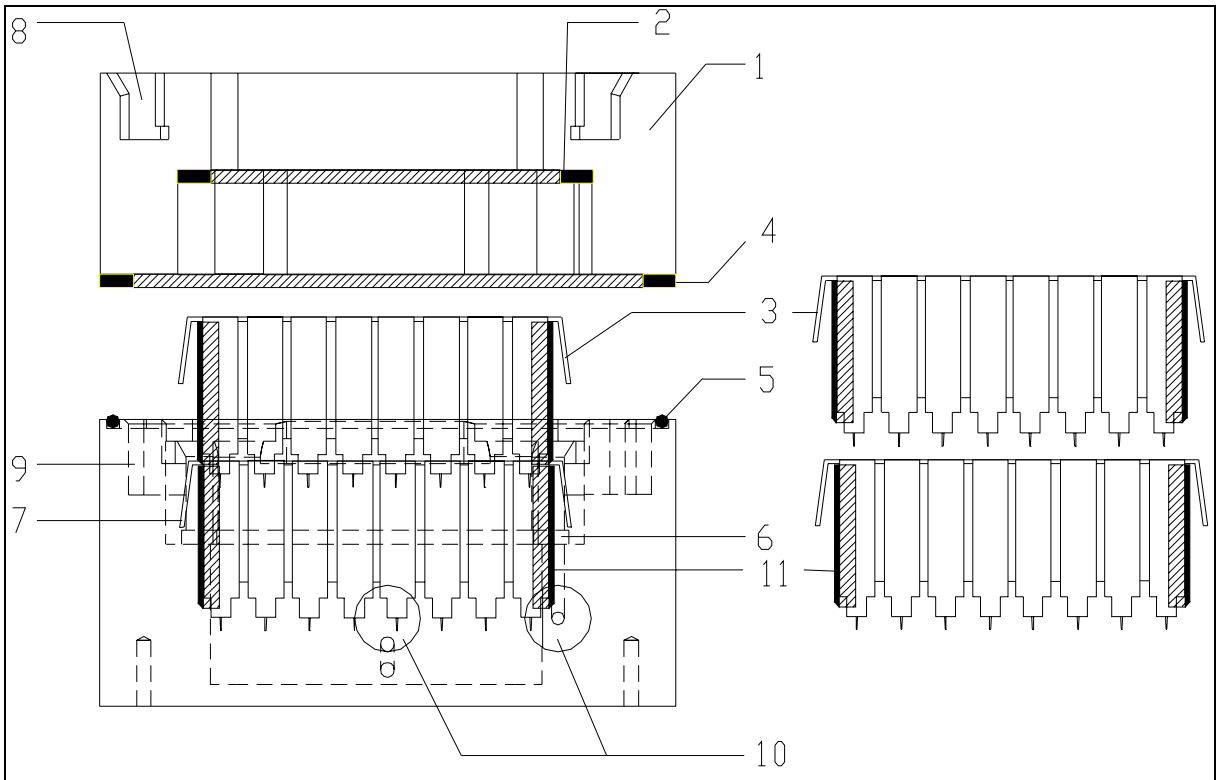


Abbildung 5-10: Querschnitt der Gesamtanordnung vor Deckelverschluß. Legende vgl. Abbildung 5-11

Die Abstandshülsen sollten zwecks einfacher Handhabung im Laborbetrieb federnd auf die vier Eckschlote aufgesteckt werden können. Sie mußten daher aus einem Material gefertigt werden, das einerseits die flexiblen Eigenschaften von Clipfedern hat, andererseits jedoch ausreichende Chemikalienresistenz und Steifigkeit aufweist. Es wurden 1.0 mm dicke Hülsen aus Kunstharz gegossen. Für eine industrielle Herstellung können diese auf ökonomische Weise auch aus Plexiglasrohren hergestellt werden. Der zum Aufstecken auf die vier Eckschlote notwendige Spalt ist ausreichend groß, um das Einstellen des Vakuums dort nicht zu behindern.

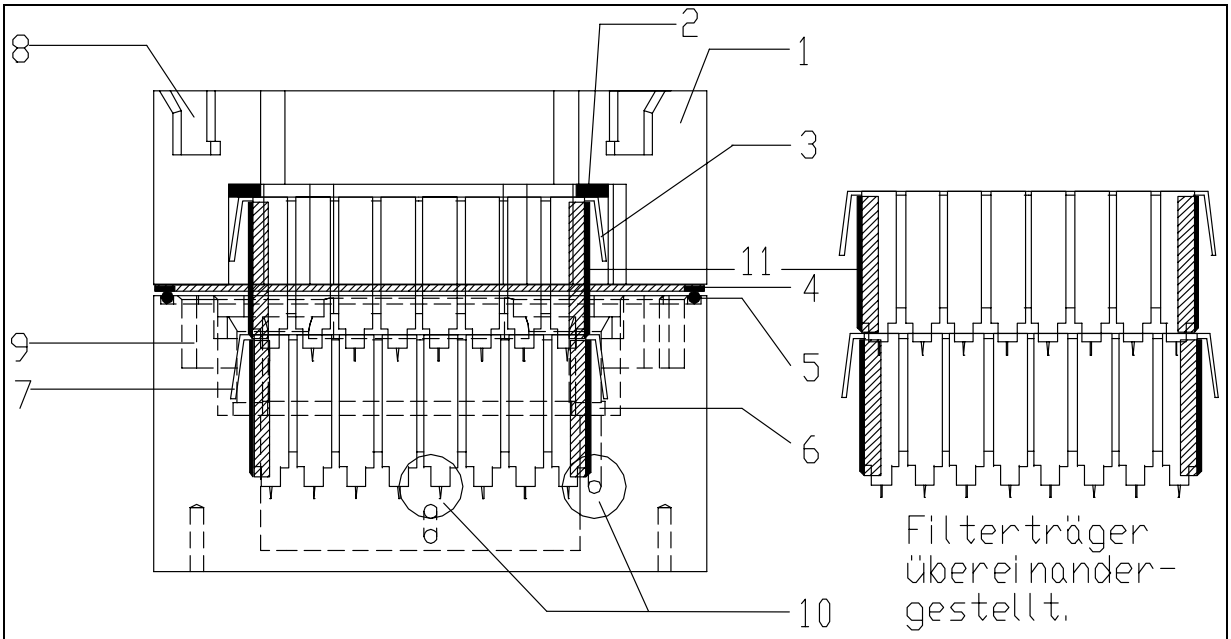


Abbildung 5-11: Querschnitt der Gesamtanordnung bei geschlossenem Deckel.

1= Deckel, 2= obere Deckeldichtung, 3= Filterträger, 4= untere Deckeldichtung, 5= O-Ring,
6= Untere Kammerdichtung, 7=Filterträger, 8= Mulden für Greifer, 9= Aussparungen Greifer, 10= Ventilanschlüsse.

5.2.3.2 Notwendigkeit von zwei unabhängig einstellbaren Vakuumbereichen

Manuell durchgeführte Präparationen mit dem Qiagen Plasmidpräparationskit „Turbo“ zeigten, daß die Filtrationsraten für die 96 Säulen durchaus unterschiedlich sein können. Die Bakterienstämme werden in speziellen Kulturgefäßen bebrütet, die sehr hochwandigen Mikrotiterplatten ähnlich sind. Nach Kultivation werden die Klone in einer Zentrifuge, in der diese Kulturgefäße eingesetzt werden können, sedimentiert. Der Überstand an Kulturmedium wird dekantiert. Es konnte nun beobachtet werden, daß die eingesetzten Klone in den 96 Kammern dieser Kulturgefäße unterschiedlich wachsen können. Ergebnis ist ein Muster unterschiedlicher Zelldichten in den gewonnenen Sedimenten. Dies schlägt sich nach erfolgter Lyse direkt im Filtrationsverhalten während der Plasmidpräparation nieder. Daher ist es günstig, wenn man bei der ersten Filtration des Lysates zunächst alle Filtrate im unteren Filter „sammeln“ kann. Zweckmäßigerweise legt man hierzu das Vakuum über einen etwas längeren Zeitraum in der oberen Ebene der Kammer an, als für die längste empirisch bestimmte Filtrationsdauer. In der vorliegenden Arbeit ist die erste Filtration nach ca. 5 Minuten beendet. Sind alle Filtrate im unteren Filter angekommen, so kann die weitere Präparation synchron fortgesetzt werden. Durch dieses Verfahren erhielt man weitaus gleichmäßigere

DNA Präparationen (DNA-Gehalt und Sequenzierbarkeit betreffend). Mit dem Qiagen Turbo-Plasmid-Präparationsprotokoll können 96 Proben in ca. 60 Minuten prozessiert werden. Die Möglichkeit, zwei getrennt voneinander regulierbare Vakuumbereiche zur Verfügung zu haben, entlastet darüberhinaus das Gesamtsystem der automatisierten Anlage erheblich. Die zeitintensiven Manipulationen zum Austausch der Filterhalter durch den Greifroboter erübrigen sich, so daß dieser in Kombination mit dem Pipettierroboter und/oder PCR-Roboter bereits weitere Aufgaben übernehmen kann, während die Filtration z.B. der Plasmidpräparation voranschreitet. Die Interprozeßkommunikation der betreiligten Software-Ansteuerungsmodule unter Windows NT (Multitasking) ermöglicht eine nahezu beliebig komplexe Logistik, so daß mehrere Prozesse oder Präparationen gleichzeitig im System ablaufen können, wenn ein geeigneter zeitlicher Versatz für die einzelnen Schritte eingerichtet werden kann. Die Hilfssysteme (PCR- oder CRS-Roboter) werden beispielsweise durch Entkopplung solcher zeitintensiver Komponenten entlastet, so daß die Anlage insgesamt für große Sequenziervorhaben ökonomischer ausgelastet werden kann. Das offene Design der Vakuumkammer ermöglicht es, auch künftige Präparationsschemata, die sich auf Adsorptionstechnologie und, oder Ionenaustausch in kleinen Säulen stützen, zu betreiben. Grundsätzlich können alle möglichen Präparationsschemata, die sich auf die Bewegung und Trennung von Substanzen beziehen, können in einem ein- oder mehrstufigen Prozeß in der Vakuumkammer betrieben werden. So sollte die Kammer immer dann in einer automatisierten Umgebung einsetzbar sein, wo Substanzen oder Teile von Substanzgemischen z.B. durch Anwendung von Über- oder Unterdruck, durch Gravitation oder Zentrifugation bewegt bzw. getrennt werden sollen. Eine Verwendung im Rahmen synthetischer Abläufe ist ausdrücklich eingeschlossen (z.B. Oligonucleotid-synthesen).

5.2.3.3 Unüberwachte Entsorgung des Abfallvolumens

Während der Plasmidpräparation von 96 Klonen fällt bei Anwendung des Qiagen Turbo-Prep ein Abfallvolumen von ca. 500ml an, das aus dem unteren Bereich der Kammer abgeführt werden muß. Der Abfall besteht aus den zur Präparation notwendigen Chemikalien und den Zellresten der Bakterien. Die Flüssigkeit ist also sowohl von ihrer chemischen, als auch ihrer biologischen Beschaffenheit nicht unbedenklich, so daß diese in einem entsprechend gesicherten Behältnis zwischengelagert werden sollte, um nach Abschluß der Präparationsarbeiten kontrolliert entsorgt werden zu können. Um diesen Ansprüchen gerecht zu werden, wurde die Vakuumkammer in ihrem unteren Kammerbereich mit einer Ansaugstelle versehen, die über ein Ableitungssystem an eine Vakuumfalle angeschlossen ist. Das während der Präparation im unteren Bereich anliegende Vakuum sorgt dafür, daß anfallende Flüssigkeit sofort in die Vakuumfalle abgesaugt wird, die gleichzeitig als deren Zwischenlager dient. Das für das untere Vakuum verantwortliche Steuerungsventil ist aus Kostengründen mit einem Messingverschluß ohne besondere Schutzauflagen ausgestattet. Wegen der Aggressivität der

Präparationsflüssigkeiten wird dieses Steuerungsventil zwischen Vakuumpumpe und Vakuumfalle gesetzt. Somit ist es auch möglich, eine Vakuumfalle aus kostengünstigem Material (Preßglas) zu wählen, weil die Gefahr einer Implosion der Flasche nicht gegeben ist. In der vorliegenden Arbeit wurde eine Öldruckpumpe (Modell RD 4, Fa. Vakubrand) mit einem Saugvermögen von 4,3 m³/h eingesetzt. Wird diese Öldruckpumpe aktiviert, dann baut sich das Vakuum vor dem Ventil und nicht in der Vakuumfalle auf, die sich bis zur Öffnung des über die elektronische Ansteuerung geregelten Ventils unter normalem Luftdruck befindet. Diese Ansteuerung ist dabei so ausgelegt, daß sich das Ventil über den Zeitraum von einer Sekunde proportional öffnet, wodurch sich das Vakuum nicht zu schnell in der Falle aufbaut. Die Vakuumbelastung für diese Falle besteht also nur während der Zeit, in der die Präparationsflüssigkeiten durch die Filtermaterialien bewegt werden. Weil in dieser Zeit der schwächste Punkt des Systems die 0.7 mm dicke Plastikschale des Filterträges ist, der direkt an den normalen Luftdruck angeschlossen ist, besteht während des gesamten Präparationsvorganges keine Gefahr der Implosion für die Vakuumfalle. Selbst wenn eine Verstopfung des Filtermaterials in allen Schloten des Filterträgers stattfinden sollte, ist die Vakuumkammer so konstruiert, daß der untere Dichtungsgummi als Sicherheitsventil fungiert: Er wird aus seiner seitlichen Phase in die Kammer gezogen und das Vakuum reißt über die auf diese Weise entstandene Öffnung zum normalen Luftdruck hin ab. Somit ist auch eine Zerstörung des Filterträgers ausgeschlossen. Um die abzusaugende Flüssigkeit möglichst rasch aus dem unteren Kammerbereich zu entfernen, wurde die Absaugöffnung direkt am Boden der Kammer angebracht. Diese ist über eine Bohrung direkt mit dem Absauganschluß verbunden. Um eine rasche Evakuierung der Gasphase zu erreichen, wurde ein Durchmesser von 3 mm gewählt.

5.2.3.4 Automatische Verdichtung des Vakuumdeckels durch spezielle Dichtungsmaterialien

Die Konstruktion der Vakuumkammer und das mit ihr ausgearbeitete Softwareprogramm machten folgende Verdichtung notwendig:

1. Das Dichtungsmaterial der unteren Kammerhälfte muß ein Material mittlerer Steifigkeit sein (60 Shore). Dies gewährleistet, daß bei der zyklischen Vernichtung des sich einschleichenden Vakuums aus den oberen Kammerbereichen die Verdichtung des oberen Filterträgers gegen den Deckel nicht gelöst wird.
2. Das Dichtungsmaterial der oberen Kammerhälfte (Deckel) muß sehr weich und elastisch sein (20 Shore). Dies gewährleistet eine ausreichende Verdichtung des Deckels gegen den oberen Filterträger sowie gegen den O-Ring der unteren Kammerhälfte. Durch seine sehr elastische Eigenschaft ist auch ein Ausgleich von Fertigungs-toleranzen des Filterträgers möglich. Die Fertigungstoleranzen des Filterträgers sind nicht zu unterschätzen, zumal diese gleich zweifach zu Buche schlagen: Zum einen ist die Auflagehöhe des unteren Filterträgers betroffen, zum

anderen auch die Abdichtung gegen den inneren Deckel an dessen Oberseite. Der O-Ring des unteren Kammerbereiches dichtet die Kammer gegen das sehr weiche Dichtungsmaterial des Deckels ab. Der O-Ring hat dabei einen Durchmesser von 2 mm. Durch seine kleine Auflagefläche ist somit ein wesentlich geringerer Anpreßdruck des Deckels gegen den unteren Teil der Vakuumkammer notwendig, um eine ausreichende Verdichtung zu erreichen. So reicht das Gewicht des Deckels zusammen mit dem sich im Inneren der Kammer aufbauenden Vakuum aus, um den Deckel mit dem von ihm eingeschlossenen Filterträger dicht zu schließen. Dieser Punkt ist für eine unbeaufsichtigte Präparation durch ein Robotersystem sehr wichtig, weil ein sich nicht ordnungsgemäß aufbauendes Vakuum die gesamte Plasmidpräparation versagen läßt. Der Abstand des O-Ringes relativ zum Dichtungsmaterial war daher in die innere Filterträger-Aufnahme des Deckels mit einzuberechnen, so daß durch leichtes Auflegen sowohl zwischen den O-Ring als auch im inneren Bereich des oberen Deckelabschlusses gegen den Filterträger verdichtet wird.

5.2.3.5 Zerstörung des Schleichvakuums durch softwaregesteuerte zyklische Ventilation

Besondere Sorgfalt wurde auf das gezielte Absaugen von Flüssigkeit aus den oberen Filterschloten in den darunter angebrachten Filterträger gelegt. Die Flüssigkeit sollte in diesem unteren Filterträger liegen bleiben, ohne sogleich weiter in den Bodenbereich der Kammer durchgesaugt zu werden. Wenn sich das Vakuum im oberen Bereich der Kammer bildet, dann wird zu dem Zeitpunkt zu dem noch keine Flüssigkeit in den unteren Filterträger getropft ist, eine geringe Menge Gas durch die unbenetzten Filterträger nach oben steigen. Somit bildet sich ein geringfügiges Vakuum im unteren Bereich der Kammer. Nachdem nun alle Flüssigkeit aus dem oberen Filterträger in den unteren gewechselt ist, wird sich im Deckelbereich der Vakuumkammer normaler Atmosphärendruck einstellen. Dadurch würde normalerweise der sich zu Beginn im Bodenbereich der Kammer gebildete Unterdruck einen Teil der nun im unteren Filterträger befindlichen Flüssigkeit auf den Boden der Kammer durchsaugen. Um diesen Effekt zu vermeiden wird das Entlüftungsventil durch die Softwareansteuerung in Intervallen geöffnet. Zu Beginn des Entlüftungsprozesses erflogt nach jeder Sekunde, nach ca. 10 Sekunden jeweils pro verstrichenen 5 Sekunden, eine volle Öffnung des Entlüftungsventils. Die Öffnungsgeschwindigkeit des Ventils ist dabei maximal. Diese zyklische Entlüftung des unteren Kammerbereiches wird, während die obere Kammerhälfte über das geöffnete Ventil an die arbeitende Vakuumpumpe angeschlossen ist, aus Sicherheitsgründen ständig beibehalten. Bei unüberwachten Roboterpräparationen könnte eine Undichtigkeit im unteren Kammerbereich zu einem ständigen Aufbau eines Unterdrucks führen. Durch die zyklische Ventilation des unteren Kammerbereiches kann dies ausgeglichen werden.

5.2.4 Das Ventilsystem

Ausgehend von einer Öldruckpumpe wird ein Vakuumschlauch über eine Y-

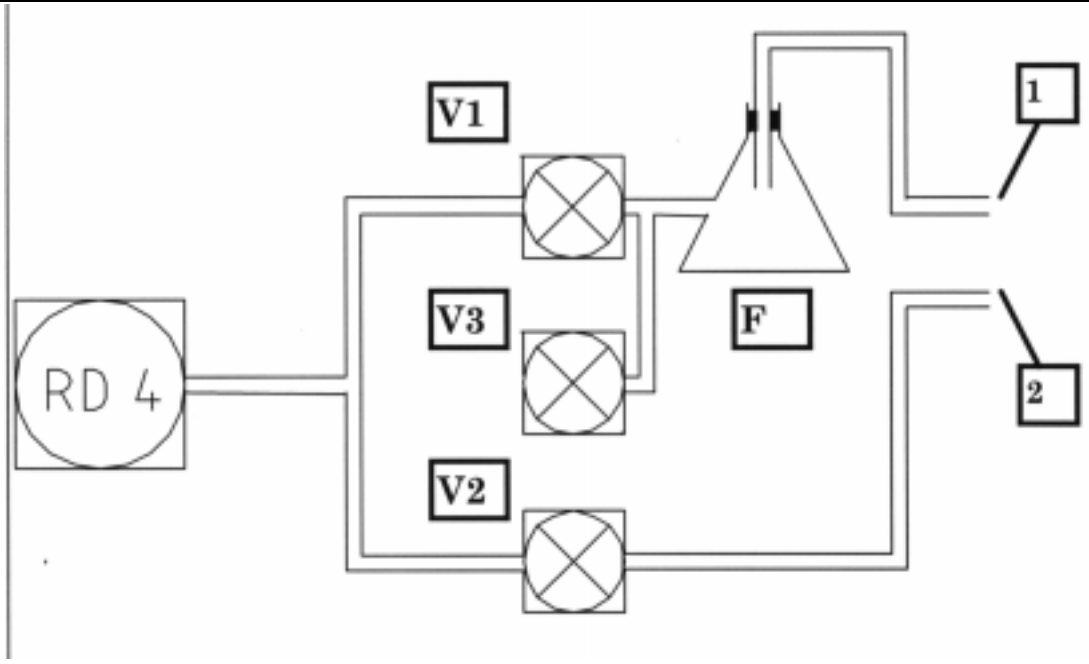


Abbildung 5-12: Anschluß der Vakuumpumpe.

1=	Anschluß an den unteren Bereich der Vakuumkammer.
2=	Anschluß an den Bereich zwischen den Filterträgern.
V1=	Elektronisch geregeltes Ventil für den unteren Kammerbereich.
V2=	Elektronisch geregeltes Ventil für den oberen Kammerbereich.
V3=	Ventil zur Zerstörung des „Schleichvakuums“ im unteren Kammerbereich.
F=	Vakuumfalle zur Aufnahme des Abfallvolumens.

Verbindung mit zwei elektronisch geregelten Ventilen verbunden, die zur kontrollierten Einstellung von zwei Vakuumbereichen in der Kammer dienen. Das Ventil, welches für den oberen Vakuumbereich zuständig ist, wird über einen Vakuumschlauch direkt mit der Kammer verbunden. Das andere Ventil, welches die Evakuierung des Bodenbereiches kontrolliert, ist über einen Vakuumschlauch mit dem Entlüftungsrohr der Vakuumfalle verbunden. Die seitliche Olive der Falle ist mit einer Schlauchverbindung an den unteren Bereich der Vakuumkammer angeschlossen. Somit kann über diese Anordnung das Abfallvolumen von ca. 500ml aus dem System befördert werden. Zwischen der Vakuumfalle und dem kontrollierenden Ventil ist über ein T-Stück das Entlüftungsventil angeschlossen, das eine wichtige Rolle bei der Aufrechterhaltung des normalen Luftdrucks im unteren Kammerbereich spielt. Es ist über sein freies Ende mit der Umgebung verbunden. Die drei Proportionalventile verfügen über eine jeweils eigene Ansteuerungselektronik [Bür1], die von der Kontrollsoftware über ein Dekodiergerät von einem Personal Computer angesteuert werden kann. Das Dekodiergerät übernimmt ebenfalls die Ansteuerung der Öldruckpumpe, wodurch die Kontrollsoftware auch dessen Aktivität kontrollieren kann. Die Ventile sind aus Ko-

stengründen einfache Gasventile ohne besondere Beschichtungen. Sie können ihr Öffnungs- und Schließverhalten über die Ansteuerungselektronik verändern. So öffnen sich die beiden Vakuumventile nach Erhalt ihres Ansteuerungssignales linear über eine Zeitspanne von 2 Sekunden zur voreingestellten Weite. Diese verzögerte Öffnung verhindert ein zu rasches Evakuieren der Vakuumkammer durch die angeschlossene Öldruckpumpe. Die Kontrollsoftware startet zunächst die Öldruckpumpe, welche daraufhin im gesamten Zuleitungsbereich zu den Ventilen einen Unterdruck erzeugt. Nach ca. 5 Sekunden wird von der Kontrollsoftware, je nach Erfordernis des aktuellen Präparationsschrittes, der Impuls zur Öffnung eines der beiden Ventile gesendet. Würde sich dieses zu rasch öffnen, so könnte der entstehende Impuls das Filtersystem oder die Vakuumkammer beschädigen. Das Entlüftungsventil, das parallel zum unteren Vakuumbereich der Kammer angeschlossen ist, öffnet sich linear über eine Zeitspanne von 0.1 Sekunden auf seine volle Weite. Es wird von der Kontrollsoftware dann in vorgegebenen Zeitintervallen aktiviert, wenn der obere Bereich der Vakuumkammer evakuiert werden soll, der untere hingegen normalen Luftdruck aufweisen muß. Dies ist der Präparationsschritt, bei dem Flüssigkeit aus den Schloten des oberen Filterträgers in den unteren gesaugt werden soll. Hier muß ein sich einschleichender Unterdruck möglichst rasch und effektiv aus dem System genommen werden (vgl.5.2.3.5 Seite 47). Weil damit zu rechnen ist, daß der Druckunterschied in diesem Fall klein ist, wird das Ventil rasch auf seine volle Weite geöffnet bzw. geschlossen. Ein sich eventuell aufbauender Unterdruck im Bodenbereich der Kammer wird durch die häufige Entlüftung auf diese Weise vernachlässigbar klein gehalten.

5.3 Weitere Konstruktionen für die Installation einer unüberwachten Plasmidpräparation

5.3.1 Verriegelbares Modulsystem

Wenn Pipettenspitzen auf die Arbeitsoberfläche des Biomek 2000 gestellt werden, ist eine besondere Verriegelung notwendig. Werden die Spitzen von der 8-fach Pipetteneinheit aus dem Halter gezogen, so kann es vorkommen, daß diese verkanten. Würde der Spitzenhalter nicht auf der Arbeitsoberfläche des Biomek festgehalten, so käme es rasch zur mechanisch bedingten Katastrophe. Der gesamte Spitzenhalter würde vom Arm des Pipettierroboters angehoben und unkontrolliert auf die Arbeitsoberfläche zurückfallen. Kommerziell ist für das in dieser Arbeit vorgestellte System keine automatisch ansteuerbare Verriegelungseinheit erhältlich. Demzufolge wurde die im folgenden beschriebene, pneumatisch angesteuerte Einheit entwickelt (vgl. Abbildung 5-13). Als modular aufgebautes System sollte auch diese Verriegelungsein-heit problemlos in das System eingesetzt, bzw. aus diesem entfernt werden können. Als besonders günstige Montageeinheit boten sich die Seitenbereiche des Biomek 2000 an, die als einsetzbare Edelstahlplatten vorliegen. Der Bereich unter diesen Platten ist hohl (Spritzgußschalen), so daß Druckluftschläuche und elektrische Leitungen auf einfache Weise untergebracht werden können. Die Ansteuerung der Verriegelungseinheit sollte über das elektronische Interface zur Vakuumkammer erfolgen. Ein elektromagnetisches Druckluftventil steuert hierbei die Zufuhr auf zwei einfachwirkende Zylinder mit Rückstellfeder (vgl. Abbildung 5-14). Diese mikropneumatischen Zylinder bewerkstelligen eine Entriegelung der von Beckman gelieferten Einstellplatten für Pipettenspitzen.

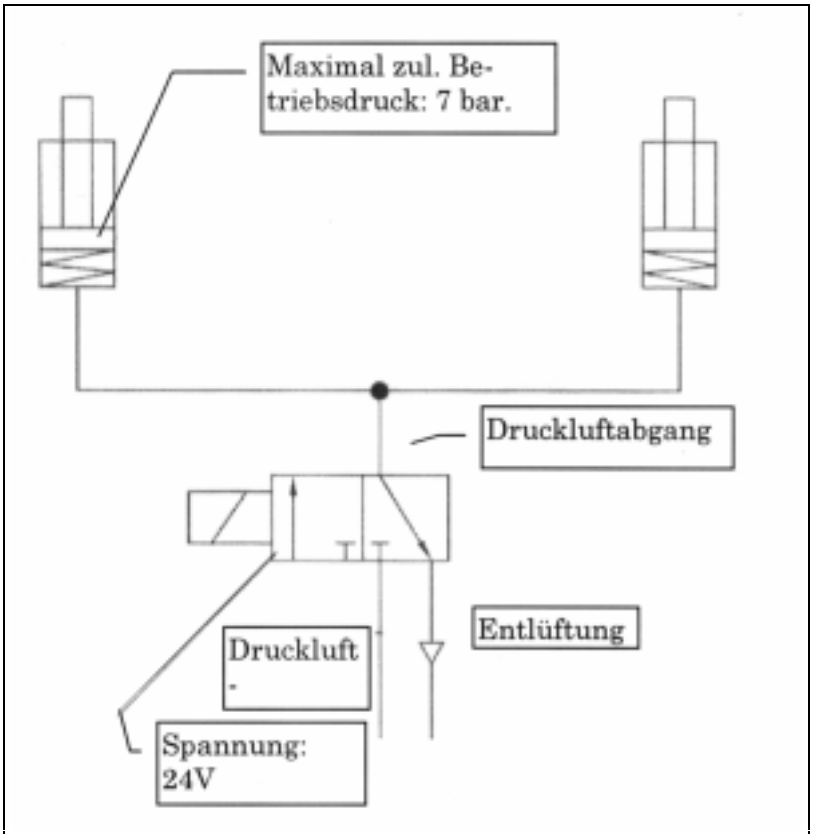


Abbildung 5-13: Pneumatik-Schaltplan für die ansteuerbare Entriegelung des Pipettenspitzenhalters am Biomek 2000.

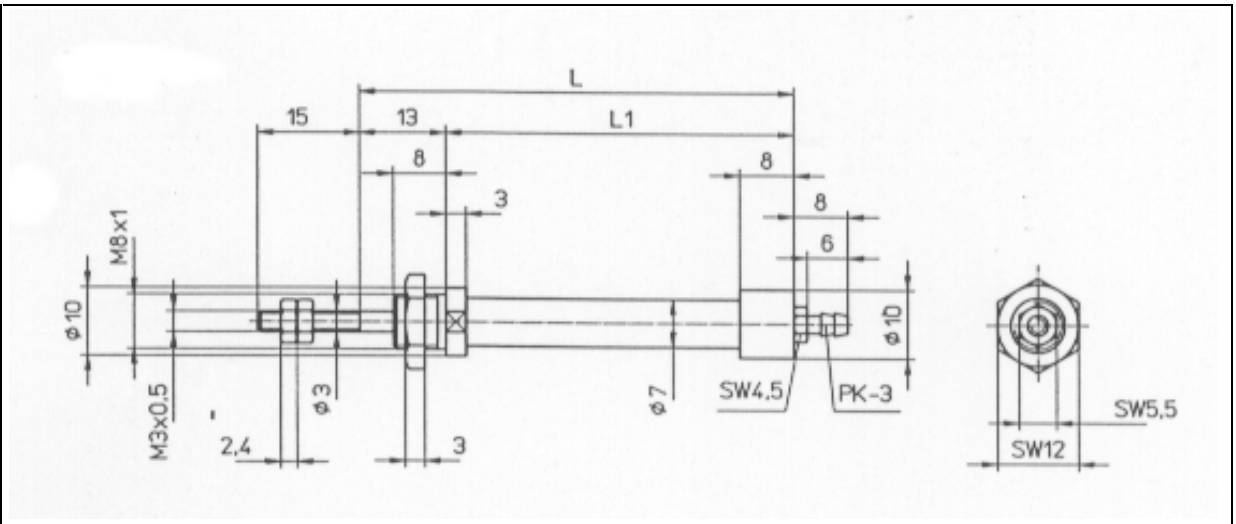


Abbildung 5-14: Einfach wirkender Zylinder mit Rückstellfeder für die ansteuerbare Entriegelung des Pipettenspitzenhalters am Biomek 2000.

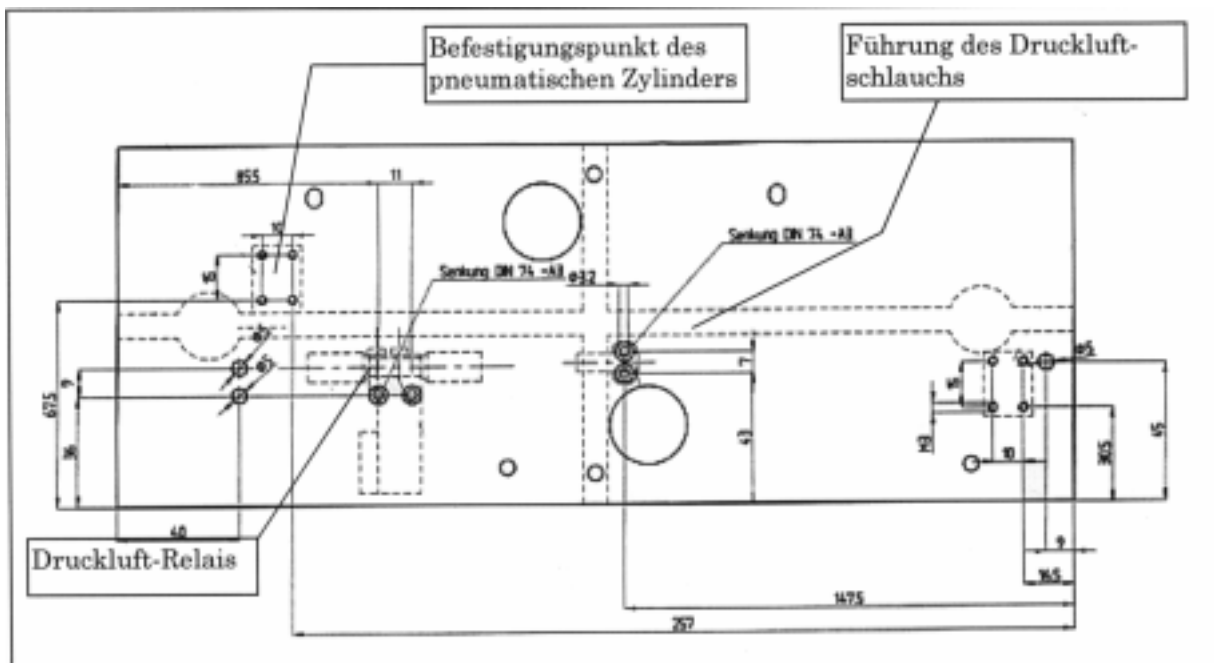


Abbildung 5-15: Gesamtansicht der Seitenplatte des Biomek 2000, welche zur Installation der pneumatischen Verriegelung wie im Bild ersichtlich modifiziert wurde.

Die Anordnung der Elemente zur elektronisch gesteuerten Entriegelung von Pipettenspitzenkästen auf dem Biomek 2000 über eine modifizierte Seitenplatte ist in Abbildung 5-15 dargestellt.

5.3.2 Ventilationssystem zur Trocknung

Das von Qiagen beschriebene Präparationsprotokoll beinhaltet das Waschen der adsorbierten DNA mit ethanolhaltigem Puffer. Versuche zeigten, daß schon Spu-

ren von Ethanol das Ergebnis der nachfolgenden Sequenzierreaktionen drastisch verschlechtern (Erläuterun-gen vgl. Kapitel 5.4.7: Ethanolproblematik des Qiaprep-Filtersystems).
Die Lösung des Problems besteht nun darin das Ethanol aus dem Bereich zwischen Auslaßdüse und Kunststoffummantelung zu entfernen. Dieser Schritt soll nicht manuell erfolgen, um die Automatisierung nicht zu unterbrechen.
Daher wurde ein Ventilationssystem eingerichtet, das erwärmte Luft (maximal 55 °C) von unten her gegen den Filterträger bläst und das Ethanol im Bereich der Auslaßdüsen verdampft.
Die Anlage zur Verdampfung der Ethanolspuren durch einen erwärmten Luftstrom besteht aus:

a) Tangentiallüfterpaar
b) Heizvorsatz
c) Luftschacht
d) Ständer für einen Filterträger
e) Elektronik zur Temperaturüberwachung
f) Computerinterface

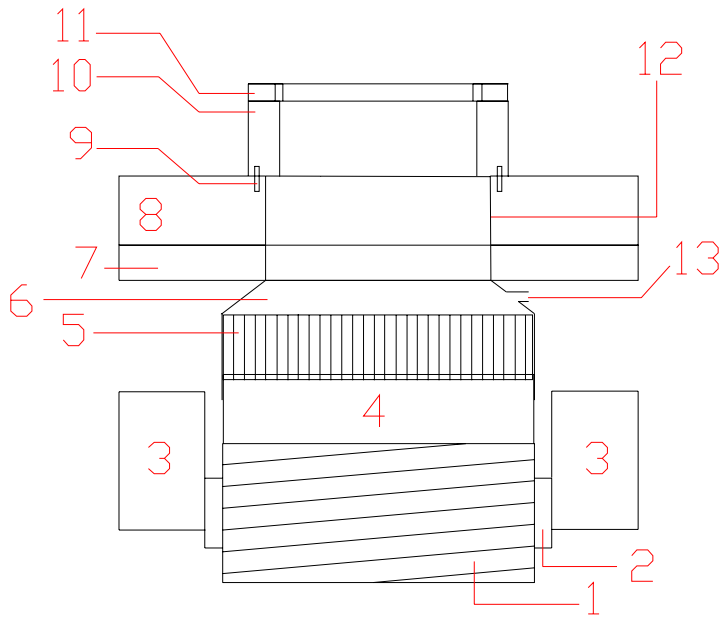


Abbildung 5-16: 1=Tangentiallüfterwalze, 2=Antriebsverbindung zum Motor, 3=Motor, 4=Luftschacht von der Tangentiallüfterwalze nach oben, 5=Düse zur Bündelung des seitlichen Luftstromes, 6= Tischplatte, 7=Acrylplatte zur Befestigung des Filterträger-Ständers, 8=Befestigungsbolzen des Ständers, 9=Abstandshalter 10=Zentrierplatte zur Aufnahme des Filterträgers, 11=Luftschacht für aufsteigenden Luftstrom, 12=Seitliche Aussparung zur Aufnahme des Temperaturfühlers

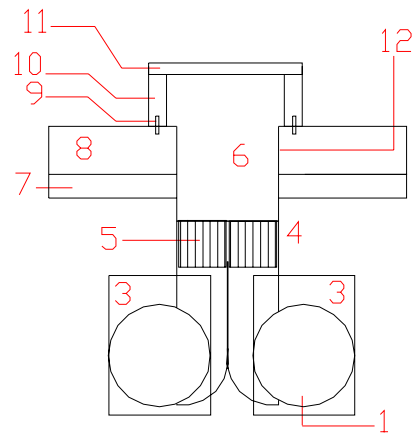


Abbildung 5-17: Legende vgl. Abbildung 5-16

Um einen möglichst homogenen Luftstrom über eine rechteckige Querschnittsfläche zu erhalten wurden zwei Tangentiallüfter an ihren Längsseiten gekoppelt, so daß sich ein gleichmäßiger Luftstrom über die gesamte Fläche des zu trocknenden Filterträgers ergibt (vgl. Abbildung 5-16, Abbildung 5-17). Die Austrittsfläche des Luftschachtes von 8cm x 12.5 cm ist

etwas größer als die zu trocknende Grundfläche des Filters. Der Luftstrom beträgt maximal 2x 80m³/h und kann über einen Vorschalttransformator über die Umdrehungsgeschwindigkeiten der Motore des Tangentiallüfterpaares geregelt werden. Der rechteckige Luftschacht ist so konstruiert, daß der Luftstrom, einer Düse gleich, zur Austrittsöffnung hin gebündelt wird. Er ist aus 2mm starkem Eisenblech gefertigt und zu einer geschlossenen Form verlötet. An dessem unteren Bereich befindet sich eine Aufnahme für die Heizung. Desweiteren sind hier die beiden Tangentiallüfter befestigt. Der obere Teil des Luftschachtes ist in einer entsprechend dimensionierten Aussparung im Arbeitstisch montiert und somit im Arbeitsbereich an genau definierter Position fixiert. Die Heizelemente und Lüfter befinden sich durch diese Anordnung unter dem Arbeitstisch. Die Öffnung des Luftschachtes ist ebenerdig mit der Arbeitsfläche des Robotertisches. Die Heizung besteht im vorliegenden Fall aus zwei Elementen, die über den Austrittsöffnungen der Tangentiallüfter befestigt sind. Jeweils zwei Multimica Lochplatten stellen die Träger für den Heizdraht dar, der in Wendeln von ca. 3mm Abstand in den Luftstrom eingespannt wird und diesen dadurch gleichmäßig erwärmt. Die Wendel sind dabei so beschaffen, daß die Umkehrschlaufe außerhalb des Luftstomes zu liegen kommt, die parallel verlaufenden Heizdrähte hingegen senkrecht zu den Multimica Trägerplatten angeordnet sind. Der Filterständer ist so konstruiert, daß er zentrierende Elemente an seinen Kontaktstellen zum Filterträger aufweist. Insbesondere sind 30° Phasen an den Berührungsbereichen zu den Abstandshülsen für die Filterhalter zu nennen. Werden die Filterträger vom CRS 465 in den Ständer eingestellt, so erfolgt ein Ausgleich von Verfahrwegstoleranzen an diesen Abstandshülsen relativ zu den gephasten Eckpunkten des Ständers. Eine erzwungene Abwärtsbewegung seitens des CRS465 richtet den Filterträger an der festen Position des Ständers aus, wobei die Abstandshülsen an den Aussparungen des Ständers ausgerichtet werden. Insgesamt erfolgt dadurch eine Zentrierung des Filterträgers in der Endlage. Der Ständer besteht aus einer oberen Trägerplatte, welche die Zentrierelemente, Aussparungen für die Abstandshülsen und die Auflagefläche für den Rand des Filterträgers bietet. An dieser Trägerplatte sind in den vier Eckpunkten 4.5cm hohe Füße befestigt, die an ihrem unteren Ende jeweils einen Stift von 0.3cm Durchmesser aufweisen. Diese Stifte fixieren den Ständer exakt über dem Luftaustrittsschacht des Gebläses, indem sie sich in entsprechende Bohrungen des Arbeitstisches fügen. Nach dem Einstellen des Filterträgers befinden sich dessen Auslaßdüsen ca. 3 cm über der Oberkante des Luftschachtes. Die Ansteuerung des Gebläses, sowie die Freischaltung der Heizung erfolgen getrennt über das Controllerinterface der Vakuumkammer. Dies hat den Vorteil, daß nach Beendigung des Trocknungsvorganges zunächst die Heizung abgeschaltet werden kann und nach zeitlichem Versatz erst die nun kühlend auf die Heizdrähte einwirkende Lüftung beendet wird. Die Heizung wird über eine Temperaturschwellwertschaltung kontrolliert. Man kann dem Gerät eine maximal erreichbare Temperatur vorgeben. Wird diese überschritten, so wird die Stromversorgung des Heizdrahtes unterbrochen. Unterschreitet die Temperatur den

vorgegebenen Maximalwert wieder, so wird die Stromversorgung des Heizdrahtes eingeschaltet. Dies ist insbesondere für DNA-Präpara-tionen wichtig, bei denen die Trocknungstemperatur bestimmte Werte nicht überschreiten darf. moderierender Software.

5.4 Modifikation der Plasmidpräparation

5.4.1 Durchmischen von Lösungen auf dem Pipettierroboter

Die Möglichkeit der Durchmischung zweier unterschiedlich dichter Medien auf einem Pipettierroboter ist sehr begrenzt. Es ist lediglich möglich, durch mehrfaches Aufziehen und Freisetzen der Lösungen mit der Multipipettenfunktion des Roboters eine Durchmischung herbeizuführen. Der eingesetzte Pipettierroboter (BIOMEK 2000) unterstützt unterschiedliche Geschwindigkeiten jeweils für den Aufzug als auch für den Auslaß der zu pipettierenden Flüssigkeit. Um Scherungen der Plasmid-DNA zu vermeiden muß diese Durchmischung relativ langsam erfolgen. Langsames Aufziehen und Enlassen der pipettierten Flüssigkeiten vermeidet ebenfalls die Kontamination der Plasmid DNA mit chromosomaler DNA während der Lyse [Qia95].

5.4.2 Wahl der Resuspensionszeiten in Puffer P_1

Die Bakterienklone wachsen in einem 96er Raster auf Agarplatten (LB-Agarplatten mit Ampilicinzusatz). Diese Rasterung macht eine rasche Überimpfung in spezielle, hochwandige Mikrotiterplatten (Beckman) möglich, die gleichzeitig als Kulturgefäße dienen. Die Bakterien werden in ampicilinhaltigem LB Medium herangezogen. Die Inkubationszeit beträgt 12-14 Stunden bei 37°C. Die Mikrotiterplatten haben ein Fassungsvermögen von ca. 3.5 ml pro Schlot, was zur Kultivierung in einem temperierten Inkubationskabinett (LabTherm, Fa. Kühner) ausreicht. Die Bakterien werden nach Erreichen der geeigneten Zelldichte (OD bei 600 nm: ca. 1.8) in einer Zentrifuge (CR 322, Fa. Jouan) bei 1800 g für 5min sedimentiert. Der Überstand wird verworfen und die Mikrotiterplatte mit ihren Bakteriensedimenten in den Regalbereich des automatisierten Systems eingestellt. Die Resuspension stellt auf einer sensorisch nicht rückgekoppelten Maschine ein Problem dar. Um die Ausbeute der Plasmidpräparation zu optimieren, ist eine möglichst vollständige Resuspension der Bakteriensedmente wichtig.
Der Resuspensionspuffer P_1 wird ohne Pipettenspitzenwechsel in die Kammern der Mikrotiterplatte eingetropft. Die Multipipette des Roboters wird dabei so hoch plaziert, daß eine Kreuzkontamination nicht erfolgen kann. Die Pipettenspitzen befinden sich ca. 2 mm innerhalb der Wände des Inkubationsgefäßes und somit ca. 2.5 cm über den Sedimenten. Es zeigte sich, daß eine Inkubationsdauer im Resuspensionspuffer von ca. 10 min. notwendig ist, um auf diese Weise eine Resuspension der Bakteriensedimente zu erhalten. In dieser Zeit sorgen unter anderem die Brownsche Molekularbewe-gung, Diffusions- und Kapillarkräfte für eine

Auflockerung des Sediments. Im Anschluß daran werden alle Proben durch langsames (ca. 0.5 cm/s) Aufziehen der Suspension und rascheres (ca. 1 cm/s) Ausstoßen derselben auf das noch existierende Sediment über die Multipipettenfunktion des Pipettierroboters resuspendiert. Um ein Verstopfen der Pipettenspitzen durch größere Sedimentfragmente zu vermeiden und den Resuspensionseffekt zu verstärken, erfolgt das Aufziehen der Resuspensionsflüssigkeit ca. 1.5 cm über dem Sediment, das Ausstoßen hingegen nur ca. 5 mm darüber. Es zeigte sich, daß nach insgesamt 10 Zyklen eine ausreichende Resuspension erfolgt ist. Die gesamte Resuspension nimmt einen Zeitraum von ca. 20 min in Anspruch.

5.4.3 Einfluß der Lysezeit im Puffer P_2 auf das Gesamtprotokoll

Erste Versuche bei der Portierung des von Qiagen vorgeschlagenen Präparationsprotokolls für Plasmide aus lebenden Bakterien scheiterten an der durch den Pipettierroboter benötigten Zeitspanne, innerhalb welcher der Lysispuffer P_2 auf die Bakterien einwirkte. Das Protokoll fordert ein Reaktionsvolumen von jeweils mehr als 200 µl für die drei Puffer (P_1 bis P_3). Der Pipettierroboter bietet jedoch nur ein maximales Volumen von 200 µl pro Spitze an, so daß jeder Schritt zweifach pipettiert werden muß. Für jede Reihe ist dann noch ein Wechsel der Pipettenspitzen notwendig, um eine Kreuzkontamination der Proben zu vermeiden. Letztendlich ergibt sich aus allen erwähnten Faktoren eine mittlere Aufenthaltsdauer der Proben von mindestens 15 Minuten im Lysispuffer. Die Resultate sehen nach elektrophoretischer Trennung auf dem Agarosegel zwar sehr vielversprechend aus (vgl. Abbildung 5-18), jedoch erweist sich die DNA in der anschließend geplanten Sequenzierung als völlig unbrauchbar, was auf ihre Denaturierung zurückzuführen ist [Bir79]. Die DNA ist gegenüber Restriktionsenzymen resistent. Bei zu langer Einwirkung von Natronlauge könnte eine zunehmende Depurinisierung der DNA erfolgen.

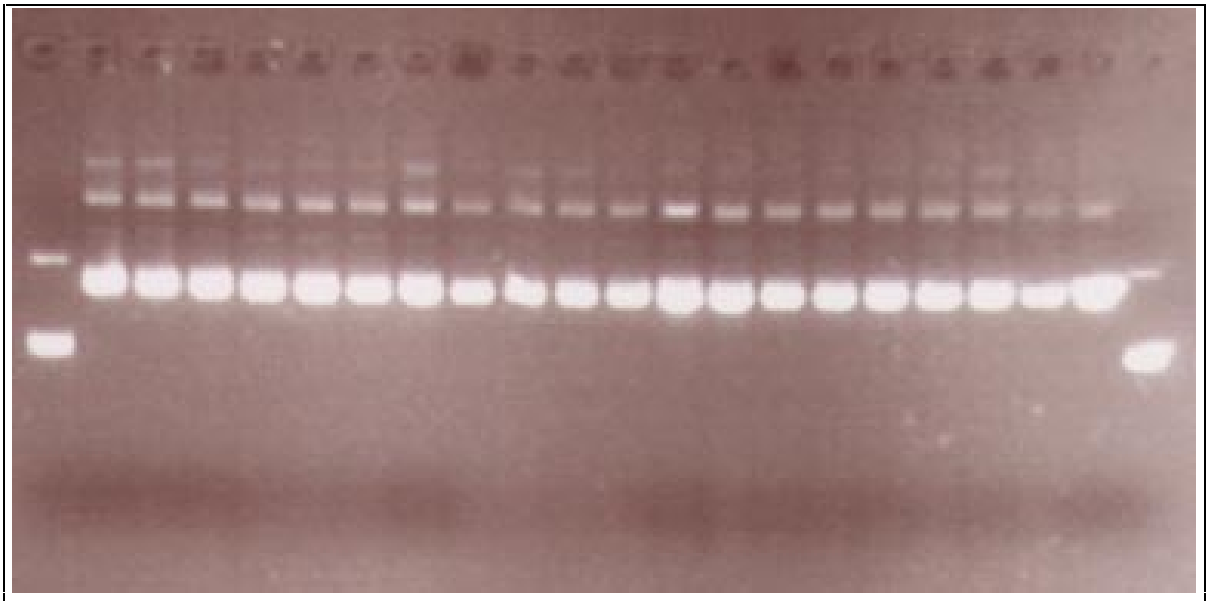


Abbildung 5-18: Ergebnis der Plasmidpräparation nach unverändertem Qiagen-Protokoll und daraus resultierender Lysezeit von 15 Minuten.

Um die Aufenthaltsdauer der Proben im Lysispuffer auf weniger als 5 Minuten zu reduzieren, wird das Volumen der eingesetzten Lösungen bei unverminderter Zelldichte jeweils unter 200µl reduziert, was einen kompletten Pippettierungsschritt einspart.

P_1:= 130 µl
P_2:= 143 µl
P_3:= 200 µl

Die Vermischung wird ebenfalls mit der Multipettenfunktion des Roboters vollzogen, beschränkt sich jedoch nur auf einen einzigen Aufzug per Probe. Auf eine explizite Verweildauer der Proben im Lysispuffer P_2 wird zu Gunsten einer verkürzten Gesamtaufenthaltsdauer vollkommen verzichtet. Dadurch daß die reihenweise Abarbeitung der Proben auch für den nachfolgenden Neutralisationspuffer gilt, liegt ein zeitlicher Versatz vor, der in etwa der geforderten Inkubationszeit von etwas weniger als 5 Minuten entspricht. Sofort nach Pipettierung des Lysis-Puffers P_2 in alle 96 Proben wird die Pipettierung des Neutralisationspuffers P_3 gestartet. Durch den zeitlichen Versatz zwischen erster und letzter Reihe wird ebenfalls auf eine explizite Verweildauer verzichtet und sofort der Transfer der Proben in den ersten Filter gestartet, um die Unterdruckfiltration einzuleiten. Durch die drastische Reduktion der Inkubationszeit der Proben im Lysispuffer verbesserte sich die Sequenzierbarkeit der über Roboter präparierten Plasmid-DNA erheblich (vgl. Kapitel 5.8 und Anhang C)

5.4.4 Der Unterdruck während des Versuchs

5.4.5 Gleicher Unterdruck im Lauf der Präparation

Der Strömungswiderstand in den Filtern kann während der Präparation unterschiedlich sein. Es gibt prinzipiell die Möglichkeit unterschiedliche Drücke anzuwenden. Ist der Widerstand hoch, dann wird stärker evakuiert, ist er niedrig, hingegen schwächer. In der vorliegenden Arbeit wurde hingegen durch unterschiedliche, empirisch ermittelte Verweildauer der Filterträger in der Vakuumkammer ausgeglichen, um schonend zu präparieren (Scherkräfte). Lediglich zum Trocknen der Filter nach dem Waschvorgang mit ethanolhaltigem Puffer wird mit erhöhtem Unterdruck evakuiert, um einen entsprechenden Gasstrom durch die Filter zu erzeugen. Alle Versuche zeigten, daß die aktuellen Präparationsschritte des Qiagenprotokolls unempfindlich sind, wenn der Flüssigkeitsspiegel in den Schloten des Filterträgers völlig durchgesaugt wurde und im Anschluß daran das Filter nur noch von Luft passiert wird. Entsprechende Tests untersuchten eine bis zu 20% ige Verlängerung über die notwendige Zeitdauer hinaus, ohne daß sich Veränderungen im Resultat ergaben. Somit werden die zur Präparation notwendigen Verweilzeiten der Filterträger in der Vakuumkammer um einen Sicherheitsbereich von 10% verlängert, um eventuellen Schwankungen in der Beschaffenheit des jeweiligen Untersuchungsmaterials vorzubeugen.

5.4.6 Variation des Unterdrucks im Lauf der Präparation

Es kann von Vorteil sein, wenn sich das auf die Filter einwirkende Vakuum während seiner Applikation variieren läßt. Ein besonderes Problem stellen Spuren von Ethanol aus dem Entsalzungsschritt dar, die im Filtermaterial zurückbleiben und eine Inhibition der an der PCR-Sequenzierung beteiligten Enzyme bewirken können (vgl. Kapitel 5.4.7). Diese Ethanolreste können aus dem Filtermaterial nur dadurch entfernt werden, indem für bestimmte Zeit ein Gasstrom durch das Filter geführt wird. Je stärker dieser Gasstrom ist, desto mehr Ethanol pro Zeiteinheit wird entfernt. So wird der ethanolhaltige Puffer während des Entsalzens mit mäßig starkem Vakuum und daher niedrigerer Durchflußrate durch das Filter bewegt. Zum Zeitpunkt der Filtertrocknung hingegen kann ein starkes Vakuum angelegt werden, das einen entsprechend intensiven Gasstrom durch das Filtermaterial ermöglicht. Über einen Digital Analogcontroller wird der Spulenstrom der entsprechenden Proportionalventile angesteuert. Seine Auflösung im Regelbereich von 0-10 Volt beträgt 16 Bit. Ein Schaltungsschritt des Controllers ist damit $1.53 \cdot 10^{-4}$ Volt. Über einen Drucksensor kann das Vakuum in der Kammer gemessen werden. Fällt der Unterdruck auf einen Wert im Bereich des Filterblindwertes, so deutet dies darauf hin, daß das Filtrat das Filter passiert hat. Somit kann die Aufenthaltsdauer der Filterträger in der Vakuumkammer niedriger gehalten werden, was den präparativen Gesamtdurchsatz der Anlage insgesamt erhöht. Eine Rückkoppelung des Systems während der Vakuumapplikation ist da-

durch möglich, daß ein Drucksensor und/oder eine optische Durchflußkontrolle mit der Regelung der betreffenden Ventilweite gekoppelt werden kann. Ein solcher Regelkreis könnte die Verweildauer der Filterträger weiter optimieren und das Verfahren insgesamt für diesen Präparationsschritt effizienter gestalten. Diese Rückkoppelung wird in künftigen Arbeiten an der beschriebenen Anlage implementiert.

5.4.7 Ethanolproblematik des Qiaprep-Filtersystems

Nachdem die Plasmid-DNA an die Filter gebunden hat, erfolgt eine Entsalzung durch zweimaliges Waschen mit jeweils 800µl 80% Ethanol pro Schlot. Im Anschluß daran wird die DNA mit H_2O bidest. eluiert. Besondere Probleme bereiteten in den nachfolgenden Sequenzierreaktionen jedoch Alkoholreste, welche sich nach dem oben beschriebenen Waschvorgang noch in der Elutionsflüssigkeit befinden können. Das Ethanol befindet sich dabei nicht nur in den Filtermaterialen des Trägers sondern insbesondere an dessen Auslaßöffnungen (zwischen den Düsen und den Umrandungen) im unteren Bereich. Die Ethanolspuren inhibieren die in den folgenden Sequenzierreaktionen beteiligten Enzyme. Zunächst wurden die Ethanolspuren durch Eindampfen und Resuspendieren der DNA entfernt. Nach dieser Prozedur konnten die gewonnenen DNA Proben erfolgreich zur Sequenzierung eingesetzt werden. In einem weiteren Versuch wurde das Vakuum im letzten Schritt des Ethanol-Waschvorgangs um 30 Minuten länger appliziert, um das Ethanol zu verdampfen. Dennoch konnte beobachtet werden, daß sich die gewonnenen Eluate gelegentlich schlecht sequenzieren ließen. In diesen Fällen konnte man, trotz der verlängerten Applikation von Vakuum, einen deutlichen Ethanolgeruch in den Proben feststellen. Der Grund lag in der besonderen Konstruktion des von Qiagen verwendeten Filterträgers (vgl. Abbildung 5-3). Im unteren Bereich ist um die jeweilige Auslaßdüse ein Kragen angebracht. Dieser verhindert eine Kreuzkontamination, wenn die Lösungen durch die Filter gesaugt werden und in den darunter befindlichen Auffänger gelangen. Ist die überwiegende Menge Flüssigkeit aus dem oberen Filterträger getropft, so kommt es oft zu lokalen Verwirbelungen, wenn ein nun stärkerer Luftstrom an den letzten noch anhaftenden Tröpfchen vorbeistreicht. Der Kragen verhindert dann effektiv ein „Verspritzen“. Allerdings ist diese Konstruktion dafür verantwortlich, daß größere Mengen ethanolhaltigen Waschpuffers in das Eluat gelangen. Ethanolhaltiger Puffer verbleibt nach dem letzten Waschvorgang im oberen Bereich des Kragens („Windschatten“) und verunreinigt bei der Elution (H_2O) die Proben: Das Eluat bildet während des Absaugens in den Eluatfänger einen Tropfen an der Auslaßdüse. Dieser Tropfen „wandert“ vor seinem Abreißen an den Düsen hoch und berührt die sich dort noch befindliche ethanolhaltige Flüssigkeit. Es findet eine Durchmischung statt. Der Tropfen reißt dann ab und fällt in den Eluatfänger. Wegen der hohen Ethanolkonzentration (80%) ist eine effektive Beeinträchtigung der bei der Sequenzierung beteiligten Enzyme gegeben. Diese Problematik wird durch Verdampfen der Ethanolspuren aus dem Kragenbereich der Filterträger

gelöst. Das Qiagenprotokoll wird in dieser Arbeit um einen Trocknungsschritt erweitert. Nach dem Entsalzen wird der Filterträger einem erwärmten Luftstrom ausgesetzt, wodurch das Ethanol effektiv aus dem beschriebenen Kragenbereich entfernt wird. Die Konstruktion des Ventilationssystems zur Trocknung ist in Kapitel 5.3.2 beschrieben.

5.5 Ansteuerung der Roboteranlage

5.5.1 Wahl des Betriebssystems

Die Vakuumkammer, wie auch alle übrigen peripheren Roboter, werden zu einer Gesamtanlage integriert. Jede dieser Einheiten bedarf einer für sie speziellen Ansteuerung und Regelung. Somit ist es notwendig für jedes dieser Module eine eigene Ansteuerungssoftware zu erstellen. Alle Ansteuerungsvorgänge sollten innerhalb dieser Steuerungssoftware geregelt werden und keiner Rückkoppelung zum aufrufenden System bedürfen. Die strenge Forderung nach Modularität bedingt, daß die jeweilige Software in sich geschlossen zu konzipieren ist. Es soll prinzipiell möglich sein, die betreffende Arbeitseinheit in ein laufendes Robotersystem einzusetzen oder aus diesem zu entfernen, ohne daß der verbleibende Rest der Anlage sich in irgend einer Weise auf die entsprechend neue Situation einstellen müßte. Somit sind beispielsweise Gerätetreiber, die für den Einsatz ihres Gerätes einen Neustart des Systems erzwingen, ungeeignet. Aus diesem Grund ergibt sich aus dem Softwareentwurf sogleich notwendig, daß das zugrundeliegende Betriebssystem sowohl das „gleichzeitige“ Nebeneinander von laufenden Programmen (Multitasking) als auch den Dialog zwischen Programmen zu deren Laufzeit (Interprozeßkommunikation) unterstützen muß. Die zu erstellende Software muß eine Schnittstelle zu externen Softwaremodulen anbieten. Über diese Schnittstelle können von der Fremdsoftware in der Steuerungssoftware (z.B. beim Start) für den Präparationsschritt relevante, variable Parameter eingestellt werden. Weil die Multitasking-Eigenschaften des DOS-Windows sich nur auf ereignis- und timergesteuerte Algorithmen beschränken, ist dieses Betriebssystem für eine Roboteranlage der geplanten Kapazität ungeeignet. Windows-NT® verfügt über ein stabiles präemptives Multitasking, das auch phasenparallele Unterprogramme (Threads) unterstützt, und es gestattet Verfahren der Interprozeßkommuni-kation einzusetzen (z.B. Dynamic Data Exchange=DDE, shared Memory). Daher erfüllt Windows-NT® als Betriebssystem alle notwendigen Voraussetzungen für die geplante Roboteranlage. Die Ansteuerungen für den Greifroboter (CRS) und der PCR-Anlage werden über serielle Schnittstellen des Typs RS232 bewerkstelligt. Für den CRS Greifroboter existiert ein DDE-Server, der ebenfalls unter Windows-NT® einsetzbar ist. Für die Modularisierung dieses Roboters muß daher ein DDE-Client geschrieben werden, der dieses Untersystem in den verbleibenden Maschinenverbund integriert. Der Mikroprozessor der „PTC-225“ (MJ Research) PCR-Maschine unterstützt die externe Übergabe von Kommandoparame-

tern, welche die Maschine über ihre serielle Schnittstelle in einen Remote-Zustand versetzt. Somit stellt dieser Roboter keine besonderen Ansprüche an das Betriebssystem, so daß er sich über eine entsprechend konstruierte Steuerungssoftware problemlos in die Anlage modular integrieren läßt. Die verschiedenen Komponenten der Vakuumkammer werden alle über ein einziges elektronisches Interface angesteuert. Zur problemlosen Decodierung wurde in diesem Interface eine parallele Schnittstelle eingesetzt. Auch deren Ansteuerung stellt an das Betriebssystem keine besonderen Ansprüche, wodurch die Vakuumkammer an ihrer Softwareschnittstelle aus einem laufenden Robotersystem entnommen bzw. eingefügt werden kann.

5.5.2 Generelle Softwaredesign-Entscheidungen

5.5.2.1 MFC 4.1

Jedes Softwaremodul, das für die jeweiligen Roboter angefertigt werden muß, soll unter Windows-NT® betrieben werden. Innerhalb der Softwaremodule findet eine enge Kooperation mit den verschiedenen Diensten des Betriebssystems statt, was von Interaktionen des Fenstermanagers, Leistungen des Dynamischen Datenaustausches (DDE, [Cla93]) bis hin zu Hardware-nahen Operationen mit seriellen bzw. parallelen Schnittstellen reicht. Das Design der Software ist rein objektorientiert und in der Sprache C++ implementiert. Um die Anbindung der Softwaremodule an Leistungen des Betriebssystems zu vereinfachen, wurden Teile des Designs von der Microsoft Foundation Class (MFC, Version 4.1) abgeleitet.

5.5.3 Ansteuerung des Pipettierroboters

Der in der vorliegenden Arbeit verwendete Pipettierroboter ist der Biomek 2000 der Firma Beckman. Er ist bereits mit einer Software zur Ansteuerung seiner Funktionen ausgerüstet. Diese Steuerungssoftware ist zum Zeitpunkt der Anfertigung dieser Arbeit ausschließlich für das Betriebssystem Microsoft® Windows, Version 3.1 ausgelegt, welches auf DOS® basiert. Die Ansteuerung des Automaten erfolgt über eine serielle Schnittstelle des Typs RS232. Mit geringfügigen Anpassungen, die Ansteuerung der seriellen Schnittstelle betreffend, wurde die Steuerungssoftware des Pipettierroboters auf das Betriebssystem Microsoft® Windows-NT® , Version 3.51 portiert. Allerdings wurden lediglich die Pipettierprotokolle mit der Ansteuerungssoftware Bioworks, Version 1.4 verwirklicht. Die übrigen Aktionen erfolgen z.T. über Interprozeßkommunikation und werden von der in dieser Arbeit geschaffenen Software moderiert. Bioworks wird somit lediglich als Server eingesetzt, um die entsprechenden Pipettierprotokolle abzuarbeiten. In einem derartigen Protokoll können Schritte enthalten sein, die nur durch die peripheren Roboter vollzogen werden können. Die notwendigen Aktionen können dann ebenfalls durch Interprozeßkommunikation verwirklicht werden.

Wenn beispielsweise ein aktueller Pipettierschritt im gerade ablaufenden Protokoll abgeschlossen ist, kann von Bioworks aus ein weiterer Prozeß im Betriebssystem angemeldet werden, der die Kontrolle des betreffenden Roboters übernimmt. Die Steuerungssoftware des Biomek 2000 unterstützt hierbei zwei Methoden. Zum einen „wartet“ der Pipettierroboter, bis der Fremdprozeß beendet wurde und nimmt dann wieder die nächst folgenden Aufgaben wahr (vgl.: Abbildung 5-19, Fall 1), zum anderen kann der Fremdprozeß „parallel“ neben dem aktuellen Pipettierprozeß ablaufen (vgl.: Abbildung 5-19, Fall 2).

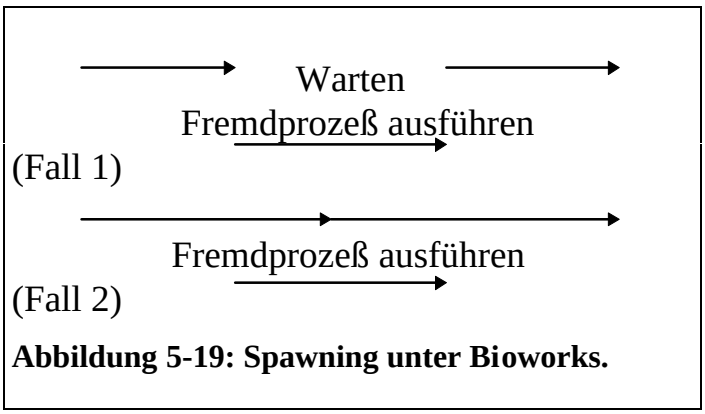


Abbildung 5-19: Spawning unter Bioworks.

Je nach aktueller Lage des gegenwärtigen Präparationsprotokolls wird die eine oder andere Methode eingesetzt. So wird aus Sicherheitsgründen stets Fall 1 implementiert, wenn der CRS-Greifroboter zur Präparation notwendiges Material aus einer peripheren Warteposition heraus auf die Arbeitsfläche des Biomek 2000 transportieren soll. Fall 2 hingegen wird beispielsweise verwendet, wenn neben einer PCR-Reaktion eine Plasmidpräparation stattfindet. Die gesamte Anlage sollte nicht deswegen stillstehen, weil ein Peripherieroboter mit einem längeren Prozeß beschäftigt ist, dessen Arbeitsbereich sich nicht mit dem Gefahrenbereich des Pipettierroboters überschneidet. Das modulare Starten von Unterprozessen erfordert die Erfüllung spezieller Anforderungen an die zu erstellenden Ansteuerungsprogramme der jeweiligen Peripherieroboter. Die Übergabe bestimmter Parameter muß dem Steuerungsprogramm des angesprochenen Peripherieroboters genügen, um die aktuelle Aufgabenstellung abzuarbeiten. Die Implementation einer DDE Schnittstelle als „Empfänger“ von entsprechenden Nachrichten aus der übrigen Roboterwelt ist notwendig, um beispielsweise eine Synchronisation verschiedener Maschinen innerhalb eines Protokolls zu erreichen. In den vorliegenden Softwaremodulen wurde die DDEML („**D**ynamic **D**ata **E**xchange **M**icrosoft **L**ibrary“) als universell dokumentierte Schnittstelle implementiert.

5.5.4 Ansteuerung der Vakuumkammer

Um die Vakuumkammer vom Rechner aus ansteuern zu können, mußte ein elektronisches Interface gebaut und die Software geschrieben werden, welche eine

Schnittstelle sowohl zu dieser Elektronik, als auch zur aufrufenden Software eingerichtet (vgl. Abbildung 5-20).

5.5.4.1 Design der die Vakuumkammer ansteuernden Hardware

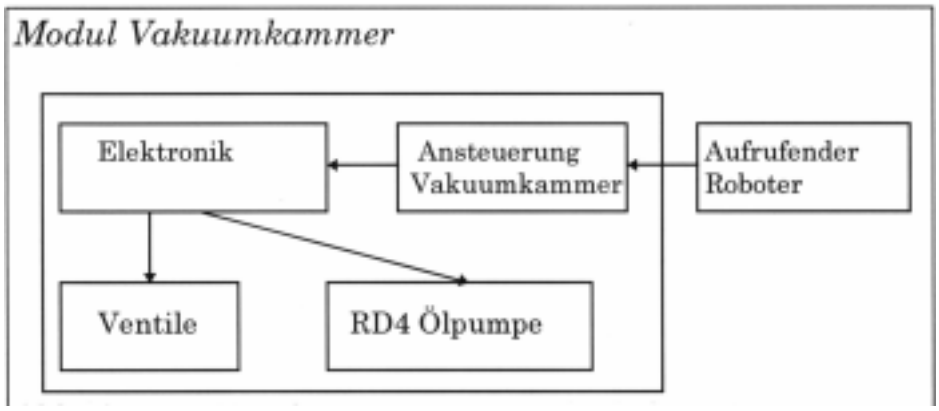


Abbildung 5-20: Schema Ansteuerung der Vakuumkammer.

Es gilt, drei Ventile und eine Öldrucksaugpumpe (RD4) von einem Computer aus anzusteuern. Die in der vorliegenden Arbeit verwendeten Ventile sind im inaktivem Zustand geschlossen, weil deren Schließer im Inneren des Ventils durch eine Feder gegen die Ventilabdichtung gedrückt wird. Der hintere Schaft des Schließers wird von einer elektromagnetisch wirksamen Spule umschlossen. Durch Anlegen von Strom an diese Spule wird der Schließer gegen die Feder zurückgedrückt. Die Ventile verfügen weiterhin über eine Elektronik, die es gestattet deren Öffnungsweite über einen variablen Spannungsbereich einzustellen. Wird die angelegte Spannung verändert, so wird über die Elektronik der Stromfluß in der Spule entsprechend reguliert. Der für die porportionale Öffnung des Ventils verantwortliche Spannungsbereich wurde in der vorliegenden Arbeit zwischen 0 Volt und 10.0 Volt festgelegt. Das Interface versorgt die Ventile sowohl mit dem notwendigen Spulenstrom, als auch über eine Spannungsteilerschaltung mit dem regelbaren Spannungsbereich von 0 Volt bis 10.0 Volt. Somit konnte die für die Präparation empirisch ermittelte Öffnungsweite des jeweiligen Ventils für den entsprechenden Kammerbereich kontinuierlich eingestellt werden. Die aufgenommene Leistung für ein Ventil ist 14 W. Es fließt ein Spulenstrom von max. 2 A. Somit wurde das zugehörige Netzteil mit einer Leistung von 120 W veranschlagt. Die ansteuernde Spannung soll laut Herstellerempfehlung aus einer anderen Spannungsquelle gespeist werden. Daher wurde eine Spannungsteilerschaltung zur der Stromversorgung der Relaisinterfacekarte eingesetzt, die von der Spulenstromversorgung der Ventile unabhängig ist. Als Schnittstelle zum Rechner wurde eine 8-Kanal Relaiskarte verwendet, die mit einem Centronics Interface ausgerüstet ist. Mit dieser Karte lassen sich alle Komponenten der Vakuumkammer über eine entsprechende Bitcodierung (siehe unten) schalten. Die Karte mußte an die Erfordernisse angepaßt werden. So weisen die Daten zur Ölvaku-

umpumpe RD4 auf einen sehr hohen Stromfluß hin, der insbesondere während der Einschaltzeit noch höher liegt. Der Schaltvorgang dieser Pumpe wurde daher auf ein Hochlastrelais gelegt, das von dem Interface angesprochen wird. Das Interface wird aus einer 12V Gleichspannungsquelle mit einer Leistung von 10.8 Watt gespeist. Die ansteuernde Software schaltet sowohl die Öldrucksaugpumpe, den Spulenstrom der Ventile als auch die über die Spannungsteilerschaltung abgegriffene Leistung erst dann zu, wenn die entsprechenden Funktionen benötigt werden. Das Gerät hat daher eine sehr geringe Ruheleistung und eine vernachlässigbare Wärmeentwicklung. Das Gerät konnte deswegen sehr kompakt gebaut werden und benötigt keine besonderen Vorrichtungen zur Kühlung seiner Komponenten. Die Einschaltzeit der Ventile ist mit weniger als 5 Minuten von kurzer Dauer, so daß auch während dieser Zeit keine nennenswerte Erwärmung innerhalb des Gerätes auftritt. Die im Gerät eingesetzte Centronics-Relaiskarte wird über eine einfache Binärcodierung angesprochen. Die acht ansteuerbaren Relais werden von 0 bis 7 durchnummeriert und als Exponent zur Basis 2 eingesetzt. Somit kann sowohl ein einzelnes Relais als auch Kombinationen derselben durch simple Addition der Codes angesteuert werden. Übergibt man der Einheit den Code Null, so werden alle Relais gelöst. In der vorliegenden Arbeit wurden folgende Ansteuerungen verwirklicht:

Relais Nr.:	1	2	3	4	5	6	7	8
Code	1	2	4	8	16	32	64	128

Relais	0	schaltet	das	Ventil 1
Relais	1	schaltet	das	Ventil 2
Relais	2	schaltet	das	Ventil 3
Relais	3	schaltet	die	Druckluftverriegelung
Relais	4	schaltet	die	Lüftersteuerung
Relais	5	schaltet	die	Spannungsversorgung für die Ventilansteuerung.
Relais	6	schaltet	die	Vakuumpumpe.
Relais	7	schaltet	die	Temperaturkontrollsteuerung

Während der Plasmidpräparation ist es erforderlich, das applizierte Vakuum zu regeln. Dies wird über eine Digital/Analog Interfacekarte mit nachgeschalteter Stromverstärkungselektronik verwirklicht. Die eingesetzte Karte ist eine DAS-1702ST [Kei97], die es ermöglicht, Spannungen im Bereich von 0 bis 10 Volt

mit einer Genauigkeit von 12 Bit (4096 Stufen) auszugeben. Die für die Ansteuerung der Digital/Analogkarte notwendige Software ist Bestandteil der in dieser Arbeit vorgestellten Ansteuerungssoftware. Die Stromabgabe der Karte ist mit 50 mA zu gering, um die Ansteuerungselektronik der Proportionalventile zu versorgen. Daher wurde eine regelbare Stromversorgung nachgeschaltet. Der erforderliche Stromkreis wurde mit Hilfe der „Electronics Workbench“ Simulationssoftware [Int97] entworfen und getestet. Kern der Schaltung ist ein Differenzoperationsverstärker (Typ: LM 324) und ein Leistungstransistor (Darlington MJ 3001). Der Operationsverstärker vergleicht die Netzteilspannung mit der Referenzspannung aus der Digital-Analog-Karte. Die Ausgangsspannung des Operationsverstärkers liegt an der Basis des Leistungstransistors an. Stimmen Netzteilspannung und Referenzspannung nicht überein, so regelt der Operationsverstärker so lange nach, bis kein Potential mehr besteht. Weil die Gesamtverlustleistung am Darlingtontransistor groß werden kann, ist dieser auf einen entsprechend dimensio-

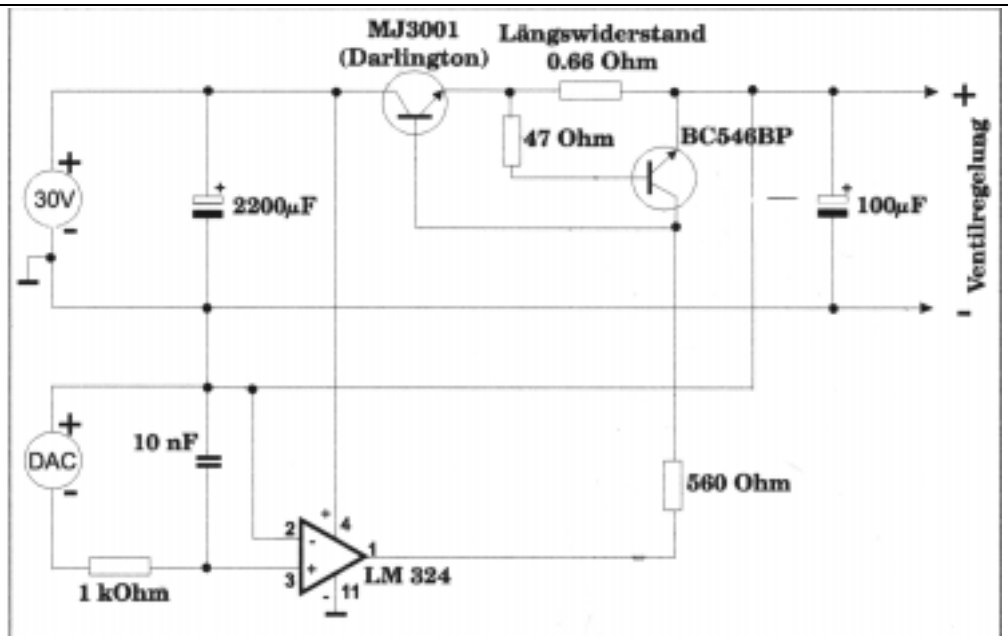


Abbildung 5-21 Verstärkerschaltung.

nierten Kühlkörper montiert worden. Die Eingangsgleichspannung von 30 Volt wird durch ein stabilisiertes Netzgerät erreicht und liegt knapp unterhalb der maximalen Betriebsspannung des Operationsverstärkers LM 324 von 32 Volt. Die Spannung am Proportionalventil kann mit einer Genauigkeit von 10 V/4096=2.44 mV geregelt werden. Der Transistor BC 546 dient zur Strombegrenzung der Ausgangsspannung. Ist der Strom größer als 1 Ampère, so fällt am Längswiderstand mehr als 0.66 Volt ab. Mit dieser Spannung schaltet der BC 546 durch und setzt das Potential an der Basis des Darlingtontransistors herab. Somit wird der Darlingtontransistor MJ3001 heruntergeregelt. Dies währt so lange, bis der Ausgangsstrom unter 1 Ampère fällt. Diese Spannungsversorgung wird an die Eingänge (vgl. Abbildung 5.21, 5.22, 5.23) V1 und V2 der Proportionalventile angeschlossen.

Belegungsplan:

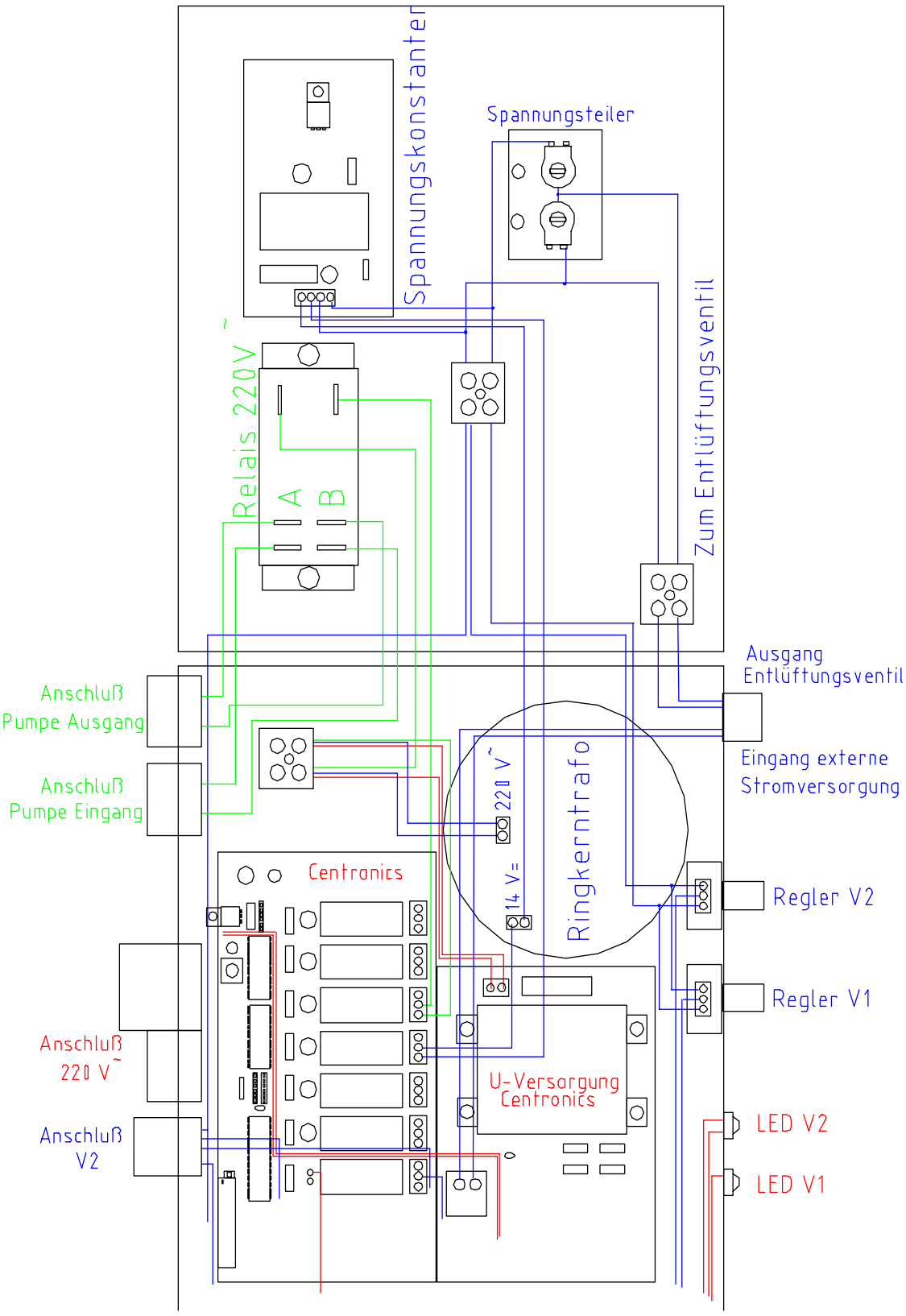


Abbildung 5-22: Belegungsplan der Ansteuerungselektronik

Schaltplan:

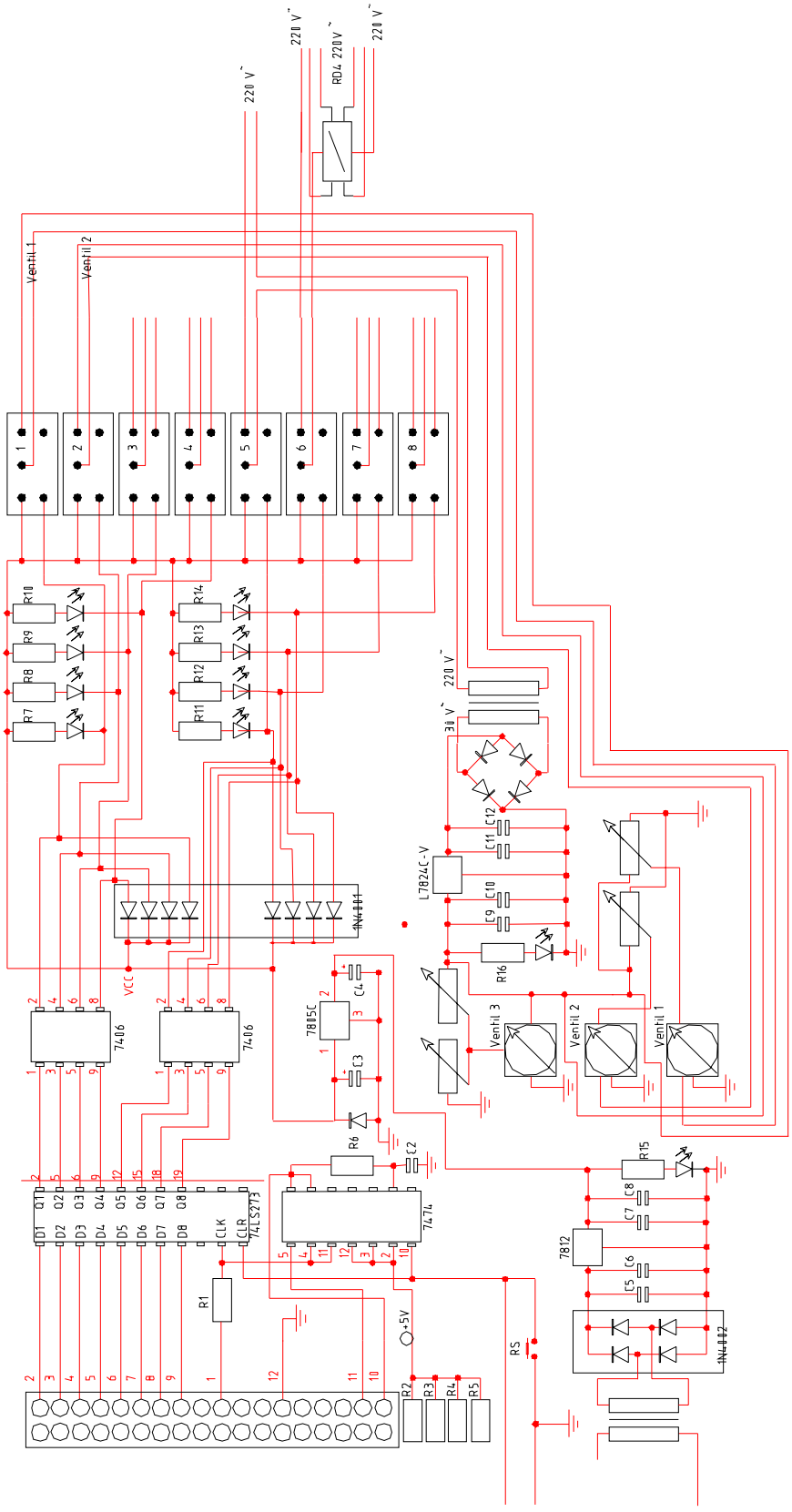


Abbildung 5-23 Schaltplan Ansteuerungselektronik der Proportionalventile

5.6 Software-Entwicklung

Software-Lösungen nicht trivialer Problemstellungen sind oft sehr komplex. Viele Fehlerquellen, die jede für sich fatale Probleme verursachen können, sind möglich. So bestand der arbeitsintensivste und zugleich komplexeste Schritt der vorliegenden Arbeit darin, eine stabile Steuerungssoftware für die automatische und unüberwachte Präparation von Plasmid-DNA aus lebenden Bakterien zu schaffen. Eigene Erfahrungen in langjährigen Projekten der industriellen Softwareentwicklung zeigten, daß die meisten Fehler während der Planung und Entwicklung der Software gemacht werden, die dann während der Implementierung zu umständlichen und fehleranfälligen Codierungen führen. Aufgrund mangelnder Dokumentation ist es oft unmöglich, die Fehlerquellen effizient aufzuspüren. So beschränkt sich die Fehlersuche in einem solchen Fall nur auf das Beseitigen von Implementierungsfehlern (beliebtestes Werkzeug ist hier der Debugger) und äußert sich durch zeitintensive Revisionen am Quellcode. Planungsfehler, welche Implementierungsfehler verursachen können, sind meist nur durch mühsame Rekonstruktion der „Planung“ erreichbar. Ein ausgezeichnetes Werkzeug, das eine Anleitung für die erforderlichen Dokumentationen während der Erzeugung von Software darstellt, ist das Modell zur Qualitätssicherung DIN (EN) ISO 9001 [ISO94] bzw. DIN (EN) ISO 9001-3 [ISO92] als Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung und Wartung von Software. In der vorliegenden Arbeit wurden Richtlinien zur **O**bjekt**o**rientierten **A**nalyse (OOA) und des **O**bjekt**o**rientierten **D**esigns (OOD) beachtet, die ausführlich in [Rum91] bzw. [Boo94] beschrieben werden. Es handelt sich um international anerkannte Verfahren, die von der Europäischen Norm ISO 9001-1994 [ISO94] vorgeschriebenen Nachweise und Forderungen für die Dokumentation des Software-Designs erbringen können. Weiterhin wurde bei der Erstellung des eigentlichen Sourcecodes (in C++) die von Microsoft eingeführte „Hungarian Notation“ [Klu88] verwendet, die über eine Typ-Dokumentation zur Stabilisierung des Codes beiträgt [Mag93]. Die originalen Kürzel der „Hungarian Notation“ wurden, basierend auf langjährigen Erfahrungen im Bereich der Softwareentwicklung, nach praktischen Gesichtspunkten modifiziert. Der Code sollte somit auch nach Portierung auf andere Plattformen (z.B. OSF-Motif) transparenter sein.

5.6.1 OOA und OOD der Ansteuerungssoftware

5.6.1.1 Objektmodell

5.6.1.1.1 Design des Objektmodells

Douglas Young, (Senior Programmer Silicon Graphics): „Identifying the precise objects or classes of objects needed to construct a program is a basic design decision that can only be made in the context of a specific problem. What objects are ultimately chosen (or „found“) is a function of the designer’s experience, preferences, and creativity. There is seldom a single right answer.“[You92]. Die Software zur Ansteuerung der in der vorliegenden Arbeit verwendeten Roboter soll so angefertigt werden, daß diese aus leicht austauschbaren und für zukünftige Anforderungen leicht modifizierbaren Bausteinen besteht, die in der Literatur auch als Software-IC’s verstanden werden [Cox91]. Ziel des Designs ist die Anforderungen der jeweiligen Ansteuerungssoftware so zu analysieren, daß die Lösung der gestellten Aufgaben in der Synthese klar definierter Software-IC’s besteht. Die **O**bjekt**o**rientierte **A**nalyse (OOA) ist der wichtigste Schritt der Softwareentwicklung. Fehlentscheidungen können zum Scheitern des gesamten Projektes führen [Bro75, Bro87, Gen91, Gui87, Jon92]. Nach sorgfältiger Analyse kann der Lösungsweg, der zu dieser Lösung führte, in einem Muster dokumentiert werden. Das Muster stellt eine Schablone dar, die einen Lösungsweg für ähnliche Problemstellungen skizziert.

> „Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so daß man diese Lösung beliebig oft anwenden kann, ohne sie jemals ein zweites Mal gleich auszuführen“. [Ale77]

Die Erstellung eines Musters folgt strengen Richtlinien, die bei [Gam95] beschrieben sind. Sie nutzt intensiv die formelle Beschreibungsgraphik von [Rum87] oder [Boo96]. Ist dieses Muster (Lösungsweg) für die Ansteuerungssoftware gefunden, so kann diese für alle Roboter in einem einzigen Schritt erstellt werden. Die für die jeweilige Maschine geltende Problematik (Hardwareabhängige Ansteuerung, Schnittstelle zum aufrufenden Programm) ist in der generellen Lösung bereits enthalten und kann durch einfache Ableitung aus ihr gewonnen werden. Nach erfolgter objektorientierter Analyse der Problematik wird das Design der Software festgelegt. Prinzipiell wird objektorientiertes Design in vier Schritten vollzogen [Gam96]:

1. Relevante Objekte aufspüren.

2. Abstraktion zu Klassen passender Granularität.
3. Schnittstellen und Vererbungshierarchie festlegen.
4. Zentrale Beziehungen zwischen Objekten festlegen.

Ist das Grundkonzept für die Problemstellung erarbeitet, so wird man dieses so lange revidieren, bis eine Software vorliegt, die wiederverwendbar erscheint. [Hic93] beschreibt einige Strategien zum Software-Entwicklungszyklus. Nach eigenen Erfahrungen ist das Design erst dann wirklich ausgereift, wenn es in unterschiedlichen Problemstellungen mehrfach wiederverwendet worden ist und sich bei dessen Implementierung keine Schwierigkeiten mehr zeigten. Wird die Benutzerschnittstelle gekapselt, so sollte eine breite Wiederverwendbarkeit über die verschiedensten Plattformen hinweg ohne besonderen Aufwand möglich sein. Die Stabilität einer Software profitiert von den Komponenten, die sich bereits in anderen Programmen bewährt haben. So ist es günstig, nicht jedes Problem mit neu zu schreibenden Softwaremodulen anzugehen, sondern auf bewährte, erfolgreich eingesetzte Software-IC's zurückzugreifen. Ein besonderer Effekt dieses Verfahrens ist, daß alle Software, die ein Software-IC bestimmten Typs enthalten, bei der Verbesserung dieses IC's automatisch mitverbessert werden. Die einfache „Austauschbarkeit" der erwähnten Software-IC's ist nur dadurch erreichbar, daß abgegrenzte Softwareeinheiten vorliegen, die ausschließlilch über genau definierte Schnittstellen miteinander „kommunizieren". Der Code innerhalb des IC's ist völlig gekapselt, d.h. alle internen Elemente der stattfindenden Datenverarbeitung sind von „außen", also von anderen Software-IC's, nicht zugänglich. Nur die Informationen welche von dem IC weitergegeben oder empfangen werden, passieren dessen Schnittstelle. Ein gutes Design achtet darauf, daß Datenelemente, auch wenn sie externen IC's zur Verfügung gestellt werden, nicht von diesen direkt verändert werden können, sondern nur über die Schnittstelle erreichbar sind (strenge Kapselung der Elemente). Nach eigener Erfahrung ist die intensive Anwendung von „Vererbung" zur Lösung von Designfragen weniger gut geeignet, weil Klassenvererbung die Kapselung aufbricht. Die internen Gegebenheiten der Oberklassen sind in den Unterklassen oft transparent. In vielen Fällen erwies sich die Anwendung von Aggregationen und Assoziationen als wesentlich effektiver, weil diese als Objektkomposition nur über die jeweiligen Schnittstellen kommuniziert. Soll ein Design verwirklicht werden, bei dem Unterklassen eine gemeinsame Schnittstelle aufweisen sollen, so ist die Klassenvererbung *das* Mittel der Wahl. Dabei überschreiben Unterklassen lediglich Operationen der (abstrakten) Basisklasse, bzw. fügen einige Operationen hinzu, um diese zu erweitern. Es erben alle Unterklassen die Schnittstelle der abstrakten Basisklasse und können auf diese Weise alle Anfragen beantworten, die an die abstrakte Basisklasse gerichtet wären. Kann man auf strenge Kapselung nicht verzichten, so kann man das Designmittel „Delegation" einsetzen. Ein Objekt verwendet hierbei ein anderes, um dort eine Anfrage abarbeiten zu lassen. Das Delegationsobjekt erhält eine Referenz auf das aufrufende Objekt und verweist damit

in der gewünschten Operation auf den Empfänger. Delegation ist auch verwendbar, um die Gesamtstruktur während der Laufzeit des Programmes zu verändern. Über die Referenz kann ein anderes Objekt übergeben werden, das bei geeigneter Implementierung anderes Verhalten zeigt als das vorherige. Somit kann eine Einheit im Zuge ihrer Weiterentwicklung (was eine simple Verbesserung sein kann) ausgetauscht werden, ohne daß der Rest der von diesem Software-IC abhängigen Software in irgend einer Weise angepaßt werden müßte. Erster Schritt in der OOA: Problembeschreibung für eine Ansteuerungssoftware. Ein externer Prozeß stößt die Ansteuerungssoftware an und übergibt ihr dabei zur Laufzeit des Programms benötigte Parameter. Diese beinhalten Informationen über die auszuführenden Aktionen, wie z.B. bei der Software zur Vakumkammer, die Einschaltzeiten der Ölunterdruckpumpe, die Ventilnummer, Zeitangaben zu dessen Öffnung usw. Diese Angaben müssen innerhalb der Software dechiffriert und in Programmparameter umgesetzt werden. Dies wird von einem sogenannten Parser übernommen. Der Parser arbeitet wie ein Übersetzer und hat wie dieser sowohl eine entsprechende Grammatik als auch einen bestimmten Wortschatz zur Verfügung. Die Kontrolle über den betreffenden Roboter wird über aufeinanderfolgende Befehle gesteuert, die in bestimmten Zeitintervallen an diesen abgegeben werden müssen. Die Steuerungssoftware muß deshalb über einen Taktgeber verfügen, mit dem eine entsprechende Synchronisation erfolgen kann. Zu Kontrollzwecken soll die Software Statusreports über den aktuellen Ausführungszustand anzeigen. Daher ist eine komplette Ausgabeeinheit für den Bildschirm notwendig. Um die aktuellen Befehle aus der Steuerungssoftware heraus an den Roboter weiterzuleiten ist eine Kommunikationsschnittstelle notwendig. Für das objektorientierte Design ist von entscheidender Bedeutung, daß, bis auf den Timer und eventuell noch die Ausgabeeinheit, alle übrigen Komponenten von der jeweilig anzusteuernden Maschine abhängig sind. Um ein wiederverwendbares Design zu erhalten, das schließlich zu Software-IC's führt muß bereits frühzeitig von der jeweiligen Maschine abstrahiert werden. Zweiter Schritt in der OOA: Abstraktion und Auffinden relevanter Objekte sowie Abstraktion zu Klassen passender Granularität. Die einzelnen Maschinen sind denkbar schlechte Objekte für das allgemeine Design, weil sie ein direktes Symbol für die Abhängigkeit sind. An deren Stelle soll ein abstrakter Roboter, oder allgemeiner noch eine „Maschine“ stehen, die in Beziehung zu einer Timer-gesteuerten Ein- und Ausgabefunktion steht. Aus der Sicht der Ansteuerunssoftware ist diese „Maschine“ jedoch nur ein Objekt, das über die vorgegebenen Parameter kontrolliert werden muß. Somit ist die best mögliche, universelle Ausdrucksform für das erste Objekt mit dem Begriff „**Controller**“ abgedeckt. Zeitabhängige Steuerungen benötigen einen Zeitgeber, der den Controller wie ein Metronom informiert. Zur direkten Information erhält der Controller zu diesem Zweck eine Schnittstelle zu dem Zeitgeber des Betriebssystems. Die Daten, welche nun Aktionen für diesen Controller definieren, betreffen direkt dessen abstrakte Welt. Wird beispielsweise für den einarmigen Greifroboter der Befehl gegeben, eine Mikrotiterplatte von A nach B zu bewegen

(was selbst ein eigenes Programm darstellt), so beziehen sich alle Koordinatenangaben auf dessen erfaßbaren Bereich. Andererseits spielen für die „PTC-225“ (M.J.-Research) PCR-Maschine Koordinaten keine Rolle. Hier zählen nur Temperatur- und Zeitangaben. Die abstrakte Maschine „Controller“ muß also über eine abstrakte Beschreibung ihrer Welt informiert werden. Sie benötigt ein Modell dieser Welt. Somit ist der Name der zweiten Klasse festgelegt: „**Model**“. Der Anwender muß zu Kontrollzwecken über den internen Zustand der abstrakten Maschine „Controller“ informiert werden. Dies soll über eine Ausgabe des aktuellen Zustandes auf den Bildschirm erfolgen. Man könnte das Objekt nun einfach „Bildschirm“ nennen, wodurch das Design jedoch unnötig festgelegt würde und erhebliches Potential verlöre. Ergeben sich Veränderungen im Modell-Datensatz, so muß sofort die Ausgabe auf dem Bildschirm an die neue Situation angepaßt werden. Dies ist jedoch nicht nur für den Bildschirm so, sondern auch für die eigentliche, anzusteuernde Maschine. Ändert sich das Modell, so sollte sich die Sicht für die Applikation ebenfalls ändern. Die allgemeine Ausdrucksform der dritten Klasse könnte mit „**View**“ bezeichnet werden. Somit ist sowohl die Bildschirmausgabe als auch die direkte Kommunikation mit der Maschine jeweils eine Variante der selben Form:

> Verändert der Controller das Modell, so werden die
> abhängigen Views darüber informiert.

Die abstrakte Maschine (Controller) muß über die neuen Anforderungen informiert werden, die von einer fremden Software beim Aufruf übergeben wurden. Diese Informationen liegen z.B. in Form einer Zeile mit Schlüsselworten vor. Diese Schlüsselworte identifizieren die unmittelbar auf sie folgende Information. Diese Information ist ebenso maschinenabhängig, wie das „Model“. In der Regel sind hier alle Informationen gegeben die das „Model“ unmittelbar betreffen. Somit ist es notwendig, einen Übersetzer für die jeweilige abstrakte Maschine einzurichten, der sowohl über die Grammatik, als auch den Wortschatz der jeweiligen Anforderung verfügt. Der Name der vierten Klasse ist mit „**Translator**“ allgemein genug gehalten. Grammatik und Wortschatz sind die beiden Elemente, welche in diesem Zusammenhang die breiteste Variation in das Design bringen. Man kann diesen Sachverhalt über ein fünftes Objekt mit dem Namen „**Language**“ widerspiegeln. Erster Schritt im OOD: Schnittstellen und Vererbungshierarchien festlegen. Die Objekte „Controller“, „Model“ und „View“ sind entkoppelt, müssen aber synchronisiert werden. Änderungen eines Objektes können sich auf andere Objekte auswirken, ohne daß das geänderte Objekt die anderen genau kennen muß. Dies ist die Definition eines „Observer-Musters“ [Gam95]. Daher bietet es sich an das Design dieser drei Objekte an das „Observer-Muster“ anzulehnen. Die „View“ Objekte sind geschachtelt. Es leiten sich gemäß eines einfachen „Composite-Musters“ zwei Views ab: „Screen“ und „Roboter“. Der „Controller“ verwirklicht die Algorithmen zur Ansteuerung der verschiedenen Maschinen (z.B. Vakuumkammer: zyklische Entlüftung) und ist somit Beispiel eines „Strategy-Musters“. Das „Translator“-Objekt wird nur wäh-

rend des Programmstartes gebraucht. Es ist nicht notwendig, daß dieses Objekt genauso lange existiert wie der „Controller“. Somit bietet sich eine einfache Assoziation an, um eine Verbindung für die Auswertung der übergebenen Information herzustellen. Das „Language“-Objekt faßt die Elemente „Grammar“ und „Vocabulary“ zusammen. Prinzipiell könnte man die Grammatik einer Sprache in Form von Methoden formulieren, die Vokabeln hingegen als deren Daten auffassen. Dies wäre die klassische Definition einer Klasse: Zusammenfassen von Daten mit den auf sie gerichteten Operationen. Allerdings würde man sich bei diesem Design die Einschränkung auferlegen, beide Elemente nicht unabhängig variieren zu können. So wäre es umständlicher, unterschiedliche Grammatiken auf den gleichen Wortschatz anzuwenden, wenn beide Elemente in ein und der selben Klasse vereint wären. Es gibt eine Reihe von Möglichkeiten, diese beiden Elemente als eigene Klassen aufzufassen und durch Vererbung in eine „Language“-Klasse zu vereinen. So könnte das „Language“-Objekt als Exemplar einer Mix-In Klasse verstanden werden (vgl. Abbildung 5-24).

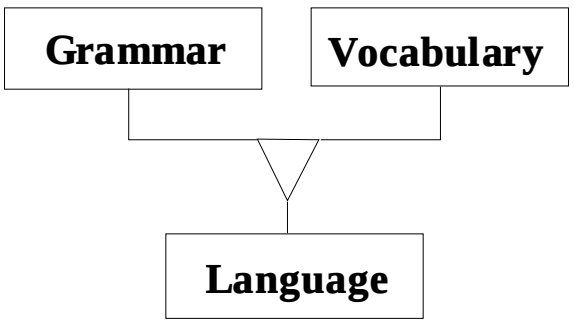


Abbildung 5-24: Mix-In-Klasse „Language“

Somit ist es möglich das „Grammar“-Objekt in künftigen Entwicklungen auszutauschen, ohne daß das „Vocabulary“-Objekt betroffen ist. Somit bleibt im gegenwärtigen Design noch die Frage offen, wie das „Trans-lator“-Objekt über sein „Language“-Objekt verfügen soll. Solange das „Translator“-Objekt existiert, benötigt es sein „Language“-Objekt. Assoziationen scheiden somit als Verbindung aus. Prinzipiell verbleiben noch die Möglichkeiten der Vererbung oder der Aggregation. Leitet man die „Translator“-Klasse von der „Language“-Klasse ab oder eröffnet man eine singuläre Aggregation (vgl. Abbildung 5-25) ? Es ist ein guter Design-Stil, die gegebene Problemstellung so natürlich wie möglich auf das gegenwärtige Abstraktionsniveau abzubilden. Vererbungen sind Elemente des OOD, um verwandte Objekte, quasie „phylogenetisch“ auseinander hervorgehen zu lassen. Aggregationen werden im OOD als Elemente der Synthese angesehen: „Das Objekt besteht aus...“, oder „Das Objekt A verwendet während seiner gesamten Existenz Objekt B für Auf-

Abbildung 5-25: Der Translator besteht aus einer Mixin-Klasse „Language“

gabe X...“. Somit wird die „Translator“-Klasse in einer singulären Aggregation mit der „Language“-Klasse verbunden.
Die Klasse „View“ ist eine abstrakte Basisklasse, in der die Schnittstelle der von ihr abgeleiteten konkreten Klassen definiert ist (vgl.Abbildung 5-26).
Dieses Design sichert die Konsistenz der Schnittstelle für alle Views. „Swatch“ ist die Klasse, deren Exemplar für die graphische Ausgabe des aktuellen inneren Zustandes auf dem Schirm verantwortlich ist. Die Klasse „RobotDevice“ kapselt die jeweilige Schnittstelle zur betreffenden Hardware. Diese Klasse ist eine Adapterklasse, welche die Hardwareabhängigkeit über eine adaptierte Klasse auflöst. Die adaptierten Klassen erhalten die Namen der anzusteuernden Geräte. Um eine maximale Flexibilität für künftige Weiterentwicklungen zu realisieren wird kein Klassenadapter sondern ein Objektadapter verwendet. Somit ist es möglich, während der Laufzeit des Programms durch einfaches Auswechseln der Objekte verschiedene Zielgeräte anzusteuern. Dies wäre durch eine starre Vererbungshierarchie im Fall des Klassenadapters (vgl. Abbildung 5-27) nicht möglich. Der Objektadapter für die beschriebenen View-Objekte verwendet folgende Objektkomposition (Abbildung 5-28): Das „Controller“-Objekt ist im vorliegenden Sofware- Entwurf das Zentrum der Applikation. Es meldet von ihr abhängige Objekte an, um sie künftig bei auftretenden Veränderungen zu informieren. Dabei wird lediglich das Signal, daß sich etwas verändert hat abgesendet. Die zugehörige Klasse verfügt über eine Schnittstelle, mit der sich die abhängigen Objekte über Anfragen mit dem neuen Zustand synchronisieren können. Die abhängigen Objekte sind für die Beschaffung der für sie relevanten Informationen selbst verantwortlich. Das „Con-troller“-Objekt

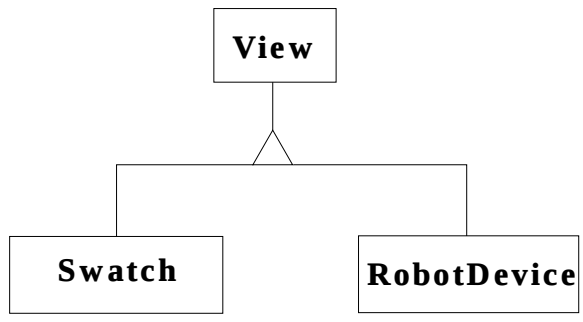


Abbildung 5-26: Ableitung der konkreten Views

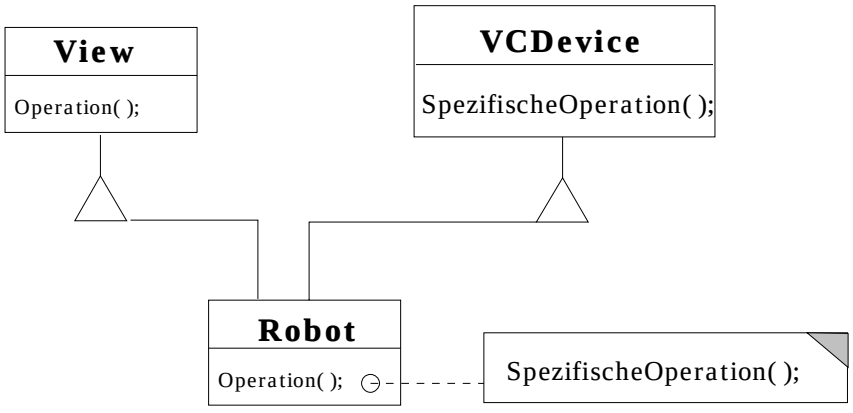


Abbildung 5-27: Klassenadapter als mögliche Lösung ist zu starr für die angestrebte Flexibilität. „VCDevice“ steht stellvertretend für alle übrigen anzusteuernden Roboter. VC= Vakuumkammer.

verwertet die Daten, welche über den „Translator“ verfügbar wurden, und arbeitet ein gegebenes Protokoll ab.

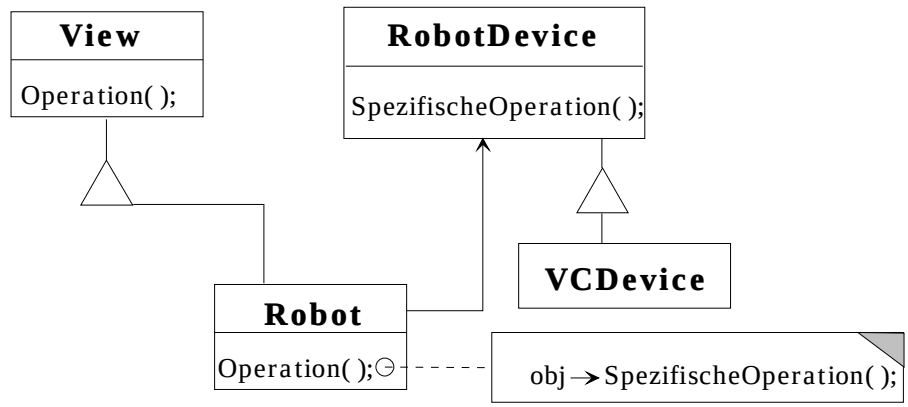


Abbildung 5-28: Objektadapter. „VCDevice“ steht stellvertretend für alle übrigen anzusteuernden Roboter. VC= Vakuumkammer.

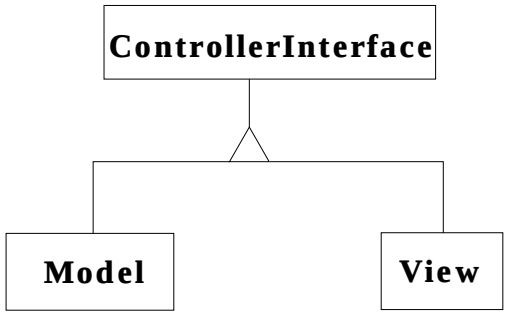


Abbildung 5-29: Interface zur Controller-Klasse.

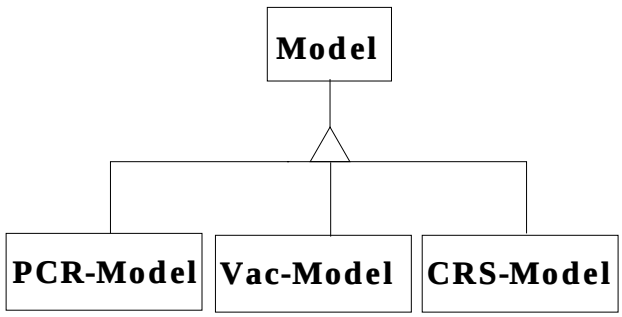


Abbildung 5-30: Vererbungshierarchie der „Model“-Klasse.

Der „Controller“ hat Zugriff auf die Systemuhr des Rechners und erhält von dieser über eine Schnittstelle ein Zeitmaß. Die „Controller“-Klasse ist abstrakt und definiert eine Schnittstelle, die von allen abgeleiteten Klassen geerbt wird. Durch die gemeinsame Schnittstelle können die von ihr abgeleiteten Klassen auf einfache Weise ausgetauscht werden. Die für jeden Roboter unterschiedlichen Methoden sind somit durch diesen Entwurf gelöst, dem ein „Observer-Muster“ zugrundeliegt (vgl. Abbildung 5-31). Das „Model“-Objekt repräsentiert die Welt der jeweiligen Maschine. Die aus dem Translator gewonnenen Daten werden in einem frühen Schritt der Programmlaufzeit über den „Controller“ in das „Model“ transferiert. Somit ist „Model“ ebenfalls ein Beobachter und wird im „Observer“ Muster als solcher angemeldet. Das „Model“ hat für diese Zwecke die gleiche Schnittstelle wie ein „View“. Man könnte nun das „Model“ von der abstrakten Basisklasse „View“ ableiten, um diese Schnittstelle zu vererben, allerdings wäre dies keine saubere Darstellung der Realität. Würde man diesem Gedanken folgen,

so würde das Design auf längere Sicht eine Fehlerquelle beinhalten, deren Auswirkungen zum gegenwärtigen Zeitpunkt nicht absehbar ist. So könnte eine Weiterentwicklung der Basisklasse „View“, ohne Berücksichtigung der von ihr abhängenden „Model“-Klasse zu ungewollten Effekten führen. Der vom Entwurf her klarere Weg ist, die reine Schnittstelle aus der „View“-Klasse zu abstrahieren und eine eigene Klasse zu definieren, von der dann sowohl „Model“ als auch „View“ abgeleitet werden. Die Klasse soll „**ControllerInterface**“ heißen (vgl. Abbildung 5-29). Das „ControllerInterface“ -Objekt garantiert die Konsistenz dieser Schnittstelle für alle Objekte, die mit dem „Controller“-Objekt interagieren sollen. Eine Weiterentwicklung des Interfaces ist durch diese Abstraktion unproblematisch geworden, weil alle von den Veränderungen betroffenen Klassen diese sogleich erben. Die „Model“-Klasse ist abstrakt und wird durch Vererbung in ihre speziellen, maschinenabhängigen Aufgaben aufgelöst (vgl. Abbildung 5-30).

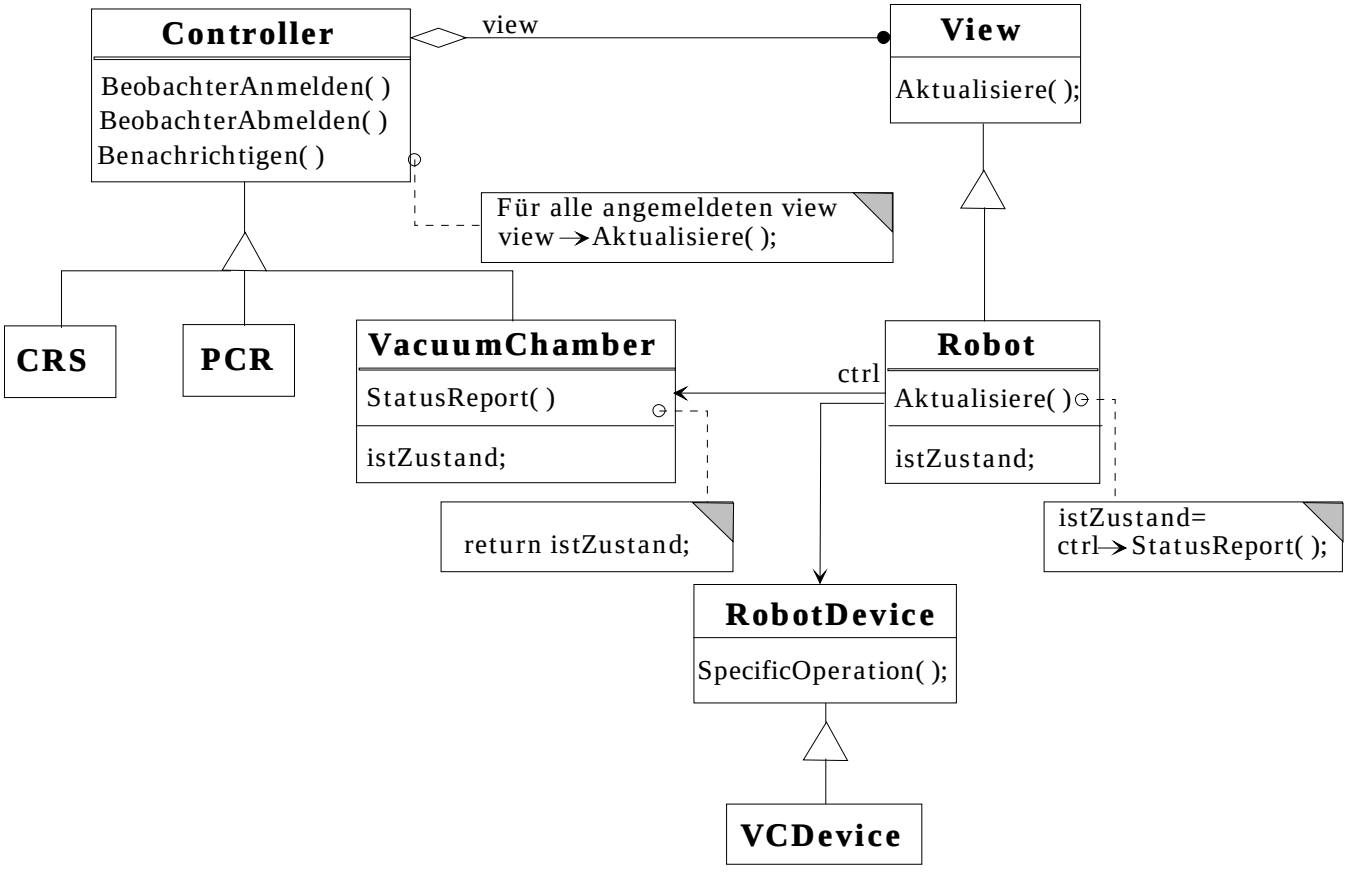


Abbildung 5-31: „Observer“-Muster für den abstrakten Roboter „Controller“

5.6.1.1.2 Diagramm des Objektmodells

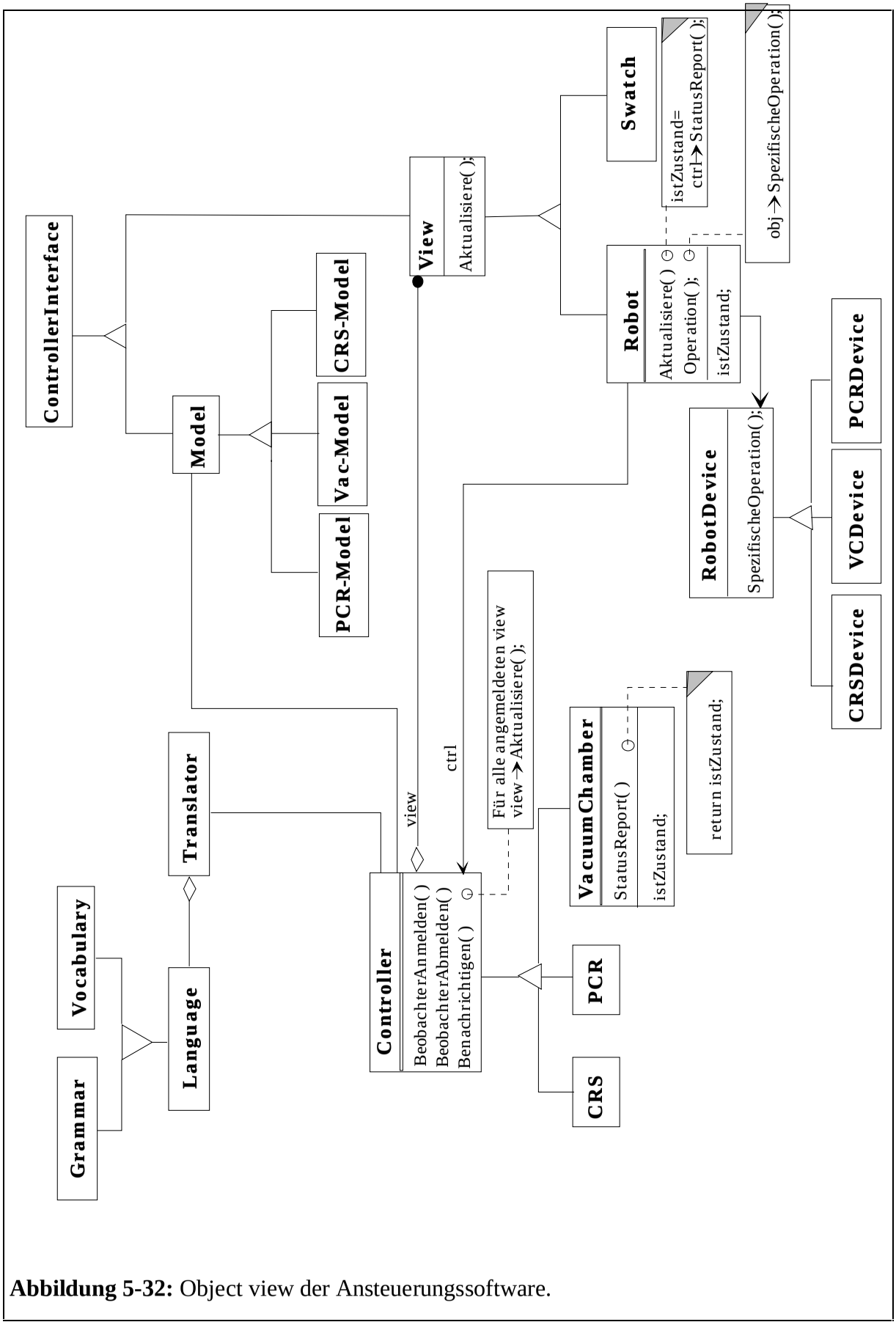


Abbildung 5-32: Object view der Ansteuerungssoftware.

5.6.2 Dynamisches Modell

Hier werden alle dynamischen, nach außen hin sichtbaren Verhaltensweisen des Systems und seiner Objekte graphisch dokumentiert. Innere Aktionen, die durch Algorithmen festgelegt sind, werden nicht berücksichtigt, sofern sie sich nicht nach außen hin manifestieren. Alle Ereignisse werden zunächst identifiziert und dann erst den Objekten zugewiesen. Auf diese Weise erhält man durch das *dynamische Modell* eine zusätzliche Kontrolle, ob das im *Objektmodell* vorgestellte Design tragfähig ist oder einer Revision bedarf. So werden zunächst Zustandsdiagramme für jedes Objekt erstellt. Hernach werden alle auf diese Weise erhaltenen Zustandsdiagramme zu einem gemeinsamen *dynamischen Modell* kombiniert. Passen alle Interaktionen reibungslos zusammen, dann ist das vorgestellte *Objektmodell* mit größerer Wahrscheinlichkeit richtig. Schritte für die Konstruktion eines *dynamischen Modells* beschreibt [Rum91]:

1. Szenarios von typischen Interaktionssequenzen vorbereiten.
2. Ereignisse zwischen Objekten identifizieren.
3. Einen Ereignispfad für jedes Szenario vorbereiten.
4. Ein Zustandsdiagramm entwickeln.
5. Ereignisse zwischen den Objekten vergleichen, um die Konsistenz zu verifizieren.

5.6.2.1 Szenarios typischer Interaktionssequenzen

Um eine Interaktionssequenz zu erhalten werden drei Schritte abgearbeitet:

1. Szenarios für den normalen Programmfluß.
2. Szenarios für Fälle, die nicht unter (1) behandelt wurden, wie z.B. Extremwerte oder Fehlberechnungen im Programm.
3. Szenarios für Fehleingaben des Benutzers oder des aufrufenden Programms.

Beispiel: Szenario für den Translator.

1. Normalfall

- Der Translator erhält vom Controller eine Zeile, die Informationen für den anzusteuernden Prozeß enthalten soll.
- Der Translator überprüft die Grammatik und den Wortschatz dieser Zeile und akzeptiert diese als Kommandozeile.
- Der Translator übersetzt den Inhalt der Zeile mit Hilfe der ihm zur Verfügung stehenden Grammatik und Wortschatz in programmrelevante Parameter.
- Der Translator gibt diese Programmparameter an den Controller zurück.

2. Extremfälle

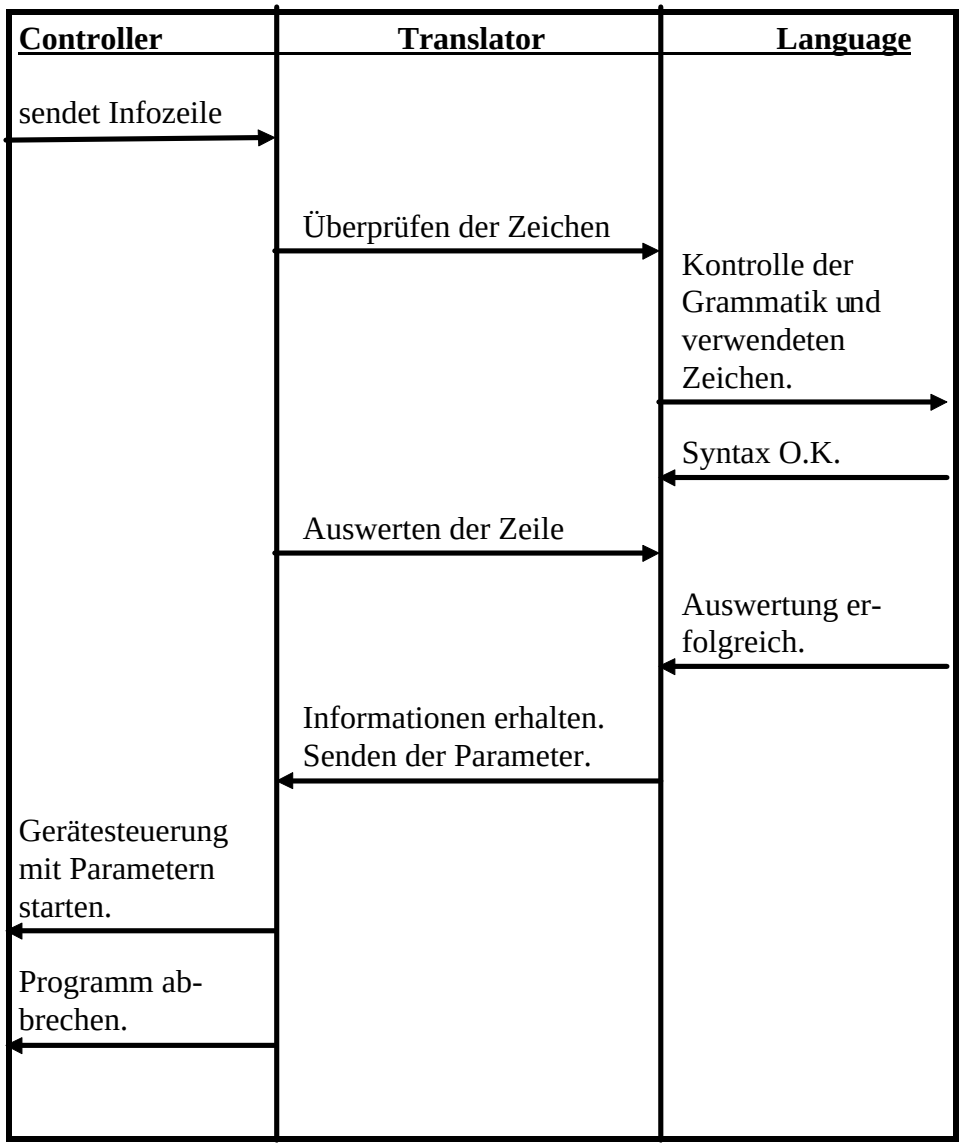
- Der Translator erhält vom Controller eine Zeile, die Informationen für den anzusteuernden Prozeß enthalten soll.
- Der Translator überprüft die Grammatik und den Wortschatz dieser Zeile und akzeptiert diese nicht als Kommandozeile, weil die Grammatik fehlerhaft ist.
- Der Translator informiert den Controller über eine fehlerhafte Kommandozeile und bricht ab.

3. Fehleingaben

- Der Translator erhält vom Controller eine Zeile, die Informationen für den anzusteuernden Prozeß enthalten soll.
- Der Translator stellt fest, daß die übergebene Zeile ungültig ist (NullPointer).
- Der Translator informiert den Controller, eine fehlerhafte Informationszeile erhalten zu haben und bricht ab.

5.6.2.2 Ereignisse zwischen Objekten identifizieren

Ereignisse, welche die gleiche Wirkung auf den Kontrollfluß haben, werden unter dem selben Namen zusammengefaßt. Die Zusammenfassungen können sich durch ihre Parameter unterscheiden. Um die Ereignisse herauszufiltern, die nur das betrachtete Objekt angehen, listet man die Szenarios in Tabellenform. Jedes beteiligte Objekt erhält eine Spalte.
Den Translator betreffen die Ereignisse:



5.6.2.3 Ereignispfad für das Szenario

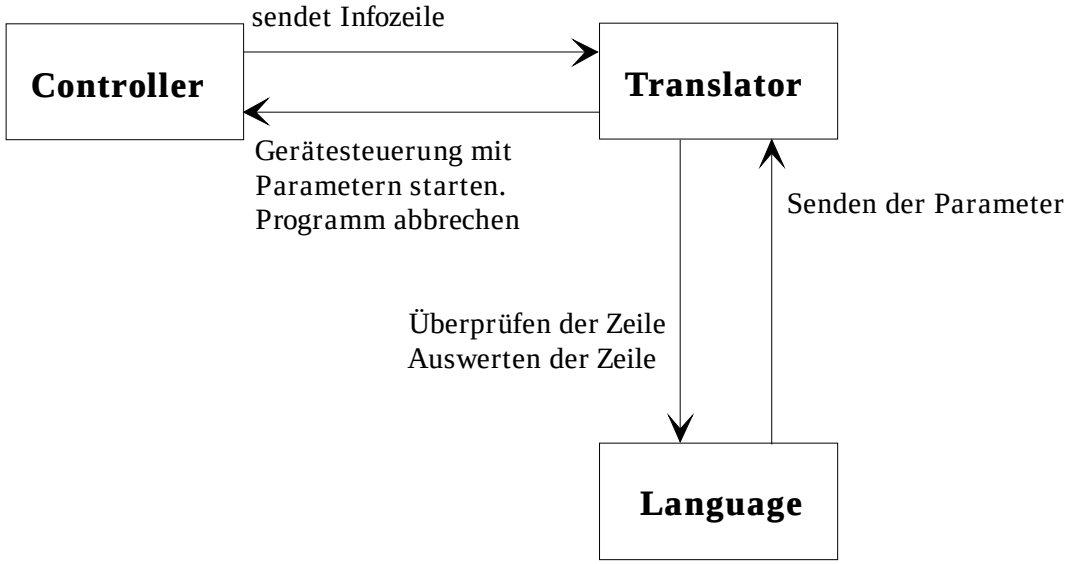


Abbildung 5-33: Ereignisflußdiagramm für das Szenario „Translator“.

5.6.2.4 Zustandsdiagramm

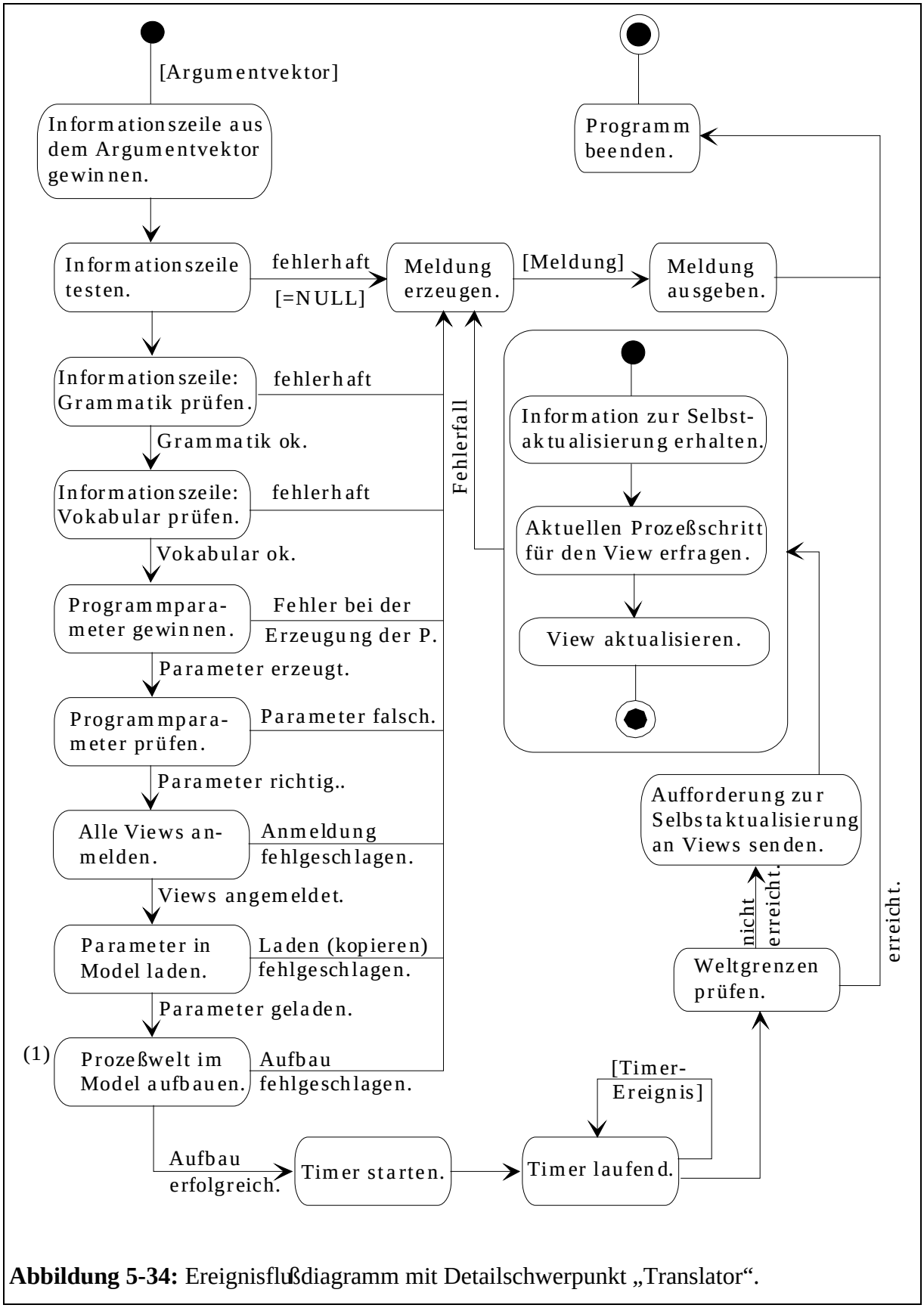


Abbildung 5-34: Ereignisflußdiagramm mit Detailschwerpunkt „Translator“.

(1):= Bedingungen, wie z.B. bei der Vakuumkammer: Die gewonnenen Programmparameter sind sTime1, sEndTime und sVorspannTime. Diese Parameter wurden aus dem Programmzeilenvektor gewonnen und in das Model geladen. Nun erfolgt der Schritt „Prozeßwelt aufbauen“: if(TimeTick < sVorspannTime) then nur Vakuum vorspannnen. if(TimeTick > sVorspannTime) && (TimeTick < sTime1)) then bVentil1 aktivieren. if(TimeTick > sTime1) && (TimeTick < sEndTime)) then bVentil2 aktivieren. if(TimeTick > sEndTime) allesEntlüften. Dies ist die „Prozeßwelt“, die dann per Anfrage abgerufen wird.

5.6.3 Funktionales Modell

Das *funktionale Modell* dokumentiert, wie Ergebnisse im Lauf der Applikation dadurch entstehen, daß auf Ausgangsdaten bestimmte Prozesse angewendet werden. Während sich das *dynamische Modell* auf den zeitlichen Verlauf konzentriert, stehen *beim funktionalen Modell* die Operationen im Mittelpunkt, die auf Daten angewendet werden. Das *funktionale Modell* wird mit Datenflußdiagrammen dargestellt.

Ein Prozeß wird im *funktionalen Modell* als Ellipse gezeichnet. In der Ellipse steht eine Bezeichnung für diesen Prozeß. Die Eingaben und Ergebnisse werden mit Pfeilen auf die Ellipse bzw. von ihr weg dokumentiert (vgl. Abbildung 5-35).

Abbildung 5-35: Symbol für einen Prozeß im funktionalen Modell. Der Datenfluß ist durch Pfeile symbolsiert.

Datenspeicher werden im *funktionalen Modell* durch dicke parallele Linien gleicher Breite symbolisiert. Zwischen diesen Linien ist die Natur des Datenspeichers angegeben (vgl. Abbildung 5-36).

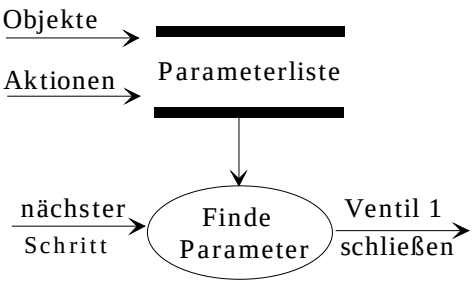


Abbildung 5-36: Symbol eines Datenspeichers.

Der Datenspeicher ist ein rein passives Element im Datenflußdiagramm, weil er nur auf Anforderungen reagiert, Daten abzulegen oder abgelegte Daten auszugeben. Die Ansteuerungssoftware verwendet einen Datenspeicher, um die Parameter im „Model“ eines Roboters zu speichern.

5.6.3.1 Ein- und Ausgabewerte identifizieren

Anhand der Problembeschreibung werden alle möglichen Ein- und Ausgabewerte identifiziert. Es kristallisieren sich all diejenigen Parameter heraus, die Daten unmittelbar bereitstellen. Alle Ereignisse, die lediglich den Kontrollfluß betreffen (z.B. Abbrechen einer Aktion), werden nicht in Betracht gezogen, weil sie keine Daten liefern. Ist eine Darstellung dieser Entscheidungsereignisse für die Übersichtlichkeit von Nutzen, so können sie dennoch im *funktionalen Modell* in Form von gepunkteten Linien (als Steuersignale) dargestellt werden. Die Ausgaben dieser Entscheidungsereignisse verändern zwar den Kontrollfluß im *dynamischen Modell*, haben aber keinen direkten Einfluß auf die Ausgabewerte.

5.6.3.2 Entwicklung von Datenflußdiagrammen

Das *funktionale Modell* wird „TopDown“ in mehreren bis vielen Schichten erstellt. Hierbei wird zunächst die oberste (gröbste) Sicht der Kommunikation modelliert (vgl. Abbildung 5-38). Am besten orientiert man sich dabei an der Systemgrenze der Applikation, weil hier an der Basis die Ein- und Ausgabeparameter am klarsten zu Tage treten (vgl. Abbildung 5-37).

Anschließend werden alle relevanten Prozesse zu Datenflußdiagrammen detailliert. Enthalten diese Datenflußdiagramme wiederum komplexere Prozesse, so können diese in einer zweiten Runde aufgegliedert werden, usw. Schließlich liegt eine detaillierte(re) Gesamtbeschreibung des Systems vor (vgl. Abbildung 5-38).

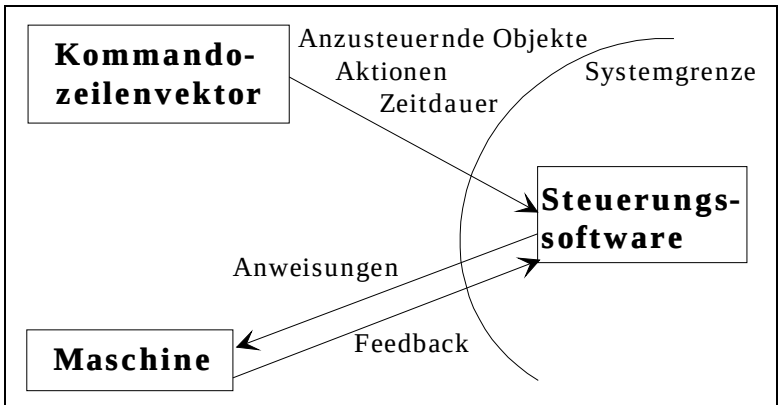


Abbildung 5-37: Erster Schritt zum funktionalen Modell.

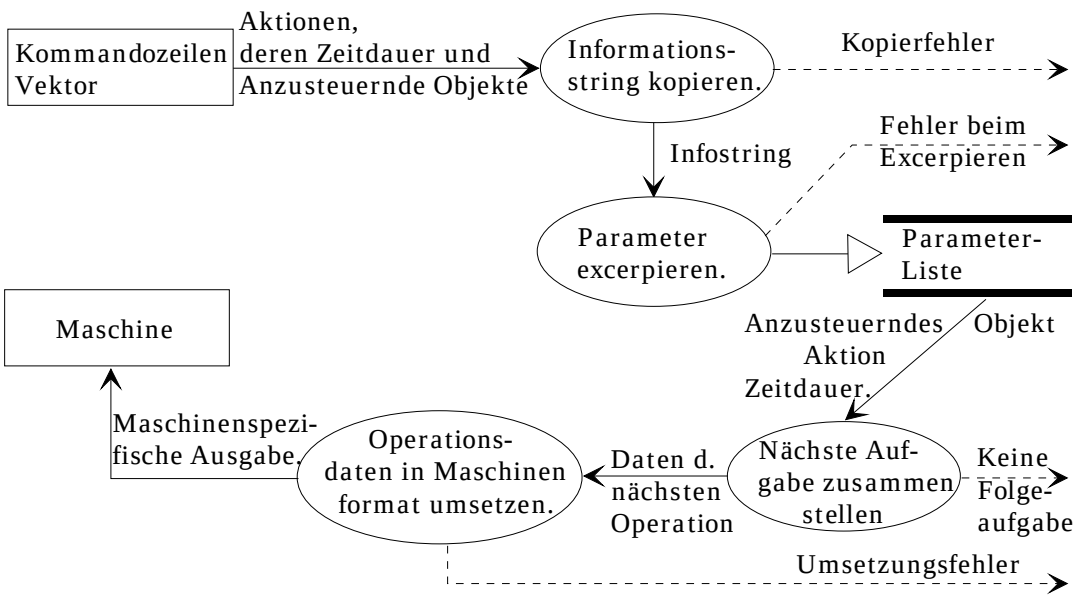


Abbildung 5-38: Oberstes Datenflußdiagramm für die Roboteransteuerung im funktionalen Modell. Die Ausnahmebedingungen bzw. Bedingungen die zum Abbruch der Applikation führen sind gestrichelt dargestellt.

5.6.3.3 Funktionsbeschreibung

In [Rum91] wird für die Funktionsbeschreibung die Formulierung von Algorithmen durch einen sogenannten Pseudocode vorgeschlagen. Diesem Gedanken wird in der vorliegenden Arbeit nicht nachgegangen, weil sich in der praktischen Programmierarbeit oft herausstellt, daß diese Pseudocodes genauso komplex erscheinen wie die eigentliche Implementierung in eine konkrete Computersprache. Durch einen detailliert beschriebenen Entwurf sind die internen Zusammenhänge einer Software durch das *Objektmodell, dynamische Modell* und *funktionale Modell* in ausreichender Deutlichkeit dokumentiert. Aus ökonomischen Gründen darf die Dokumentationsarbeit, den Entwicklungsprozeß jedoch nur so weit belasten, daß der gewonnene Nutzen den Aufwand rechtfertigt. Langjährige Praxis bei der industriellen Softwareerstellung zeigte, daß sich eine Funktionsbeschreibung am effizientesten in den Quelltext der Applikation integrieren läßt. Außerdem kann durch dieses Verfahren eine automatisierte Dokumentation der Funktionsbeschreibung ermöglicht werden, was den Forderungen des ISO9000, Teil 3 (§§ 4.2.1; 4.2.2; 4.4.b,c,e; 5.4.2.3;5.4.5,d)-Verfahrens [ISO92]entgegenkommt. Hierzu sind einige Konventionen notwendig, um diese Aufgabe an eine automatisch arbeitende Dokumentationssoftware zu übertragen. Gemäß ISO9000, Teil 3 wird die Planung des Qualitätssicherungssystems in §§ 5.4.2.3; 5.6.1; 5.6.3; 6.1.1,a,b,c,d,f; und 6.2.2,a,c und 6.5 weitgehend vorgegeben. In der Praxis hat sich folgende Unterteilung der Quellcodedokumentation bewährt:

1. Jedes Modul wird in einen klassenbeschreibenden „Header“ und einen funktionsauflösenden Rumpf unterteilt. Der „Header“ darf keine Funktionalität aufweisen.
2. Zu Beginn der Dateien befindet sich ein beschreibender Kommentar mit folgender Struktur:

```
/*~------------------------------------------------------------*/
/* Klassenname: Kurzbeschreibung der Hauptaufgabe             */
/* Name der Datei                                             */
/* Ableitungsebene                                            */
/* Detaillierte Beschreibung der Hauptaufgaben                */
/* Entwicklungsumgebung.                                      */
/* Graphische Schnittstelle.                                  */
/* Betriebssystem.                                            */
/* Autor.                                                     */
/*------------------------------------------------------------*/
/* Regeln für das Layout.                                     */
/* Abänderung hinsichtlich der Hungarian Notation.            */
/*------------------------------------------------------------*/
/* Entwicklungsgeschichte dieses Moduls:                      */
/* Datum      Grund für die Änderung  Versionswechsel.        */
/* Aktuelle Gesamtversion.                                    */
/*------------------------------------------------------------~*/
```

In dieser Kommentarsektion sind beispielsweise die Modifikationen zur „Hungarian Notation“ dokumentiert, so daß der Leser des Quelltexteden Schlüssel für die Typisierung erhält.
3. Jede Funktion erhält einen beschreibenden Kommentar mit folgender Struktur:

```
/*~------------------------------------------------------------*/
/* Ausführliche Beschreibung der Funktion                     */
/*------------------------------------------------------------*/
/* Fehlersituationen.                                         */
/*------------------------------------------------------------*/
/* Verhalten der Funktion.                                    */
/*------------------------------------------------------------*/
/* Verbesserungsvorschläge.                                   */
/*------------------------------------------------------------*/
/* Entwicklungsgeschichte dieser Funktion:                    */
/* Datum   Grund für die Änderung   Version der Funktion.     */
/*------------------------------------------------------------*/
/* Rückgabewert der Funktion.                                 */
/*------------------------------------------------------------~*/
```

4. Jede Zeile innerhalb der Funktion wird hinsichtlich ihrer Teilaufgabe (nicht ihres syntaktischen Inhaltes) kommentiert. Diese Beschreibung konzentriert sich auf die erreichte Funktionalität und nicht auf das Implementierungsproblem. Die automatisch arbeitende Dokumentationssoftware kann die geeigneten Kommentarsegmente anhand der Zeichenkombinationen /*~ ~*/ erkennen. Die entsprechenden Zeilen können durch Parsen der Schlagworte identifiziert werden. Somit kann der aktuelle Stand der Entwicklungsarbeit excerpiert werden. Durch die oben beschriebenen Punkte 1-4 ist eine hochauflösende Funktionsbeschreibung sowohl der Klassen, als auch der Implementation gegeben.

5.6.3.4 Einschränkungen feststellen

Funktionale Einschränkungen zwischen Objekten sind dadurch gekennzeichnet, daß mindestens zwei Objekte in eine übergeordnete Abhängigkeit geraten. Werden die Einschränkungsgrenzen überschritten, so ergibt sich in der Regel ein Ausnahmefall. Für die Geräteansteuerung liegen funktionale Einschränkungen beispielsweise für folgende Objekte vor: Die Größe des Vokabulars muß n > 0 Elemente enthalten, damit der Translator von dem Language-Objekt Gebrauch machen kann. Die Sprache muß aus einem Vokabular und der dazu passenden Grammatik bestehen, damit der Translator angewendet werden kann. Die erzeugten Operationsdaten dürfen nicht die Grenzen der Prozeßwelt überschreiten.

5.6.3.5 Optimierungen

Die Optimierungen ergeben sich aus den jeweiligen Möglichkeiten der anzusteuernden Maschinen. So ist es bei dem Hilfsroboter CRS 465 möglich, ganze Unterprogrammteile, die in RAPLII verfaßt wurden, in die RISC-Workstation C500 zu laden und durch die beschriebene Ansteuerungssoftware anzustoßen. Dies ermöglicht eine sehr schnelle Kommunikation mit diesem Roboter und verkürzt Reaktionszeiten für diese Maschine. Die PTC 225 PCR-Maschine verfügt über eine Schnittstelle zu deren intern programmierbarem Prozessor, so daß deren Protokolle auch der Ansteuerungssoftware zur Verfügung stehen. Somit können manuell eingegebene Labormethoden, die unter entsprechenden Namen abgespeichert sind, auch von der Ansteuerungssoftware genutzt werden. Erfolgt eine Veränderung der Protokolle in der Maschine im Rahmen einer Laboroptimierung, dann stehen diese ebenfalls dem automatisierten System zur Verfügung.

5.7 Präparation anhand chromosomalen Materials von *Arabidopsis thaliana*

Die automatisierte Präparation von Plasmiden und deren anschließende PCR-Sequenzierung mit der in dieser Arbeit beschriebenen Anlage wurde innerhalb des *Arabidopsis*-Genomsequenzierprojektes an der GBF getestet. In diesem Projekt wird unter anderem das Chromosom Nummer 4 per Shotgun-Sequenzierung [Ban83] analysiert. Die so gewonnnenen Bruchstücke liegen in BACs (Bacteria Artificial Chromosomes) einkloniert vor. Aus den BAC-Inserts werden kleinere Fragmente erzeugt und in Plasmid-Vektoren (Träger-organismus: *Escherichia coli* XL-1blue) amplifiziert. Diese Plasmid-Vektoren werden über das modifizierte Qiagenprotokoll auf der beschriebenen Roboteranlage präpariert und zur PCR-Sequenzierung verwendet. Die im Folgenden vorgestellten Daten sind mit einem ABI-Sequenzer (Modell 377) gewonnen worden. Die Qualität der Ergebnisse läßt sich mit denen von manuell präparierten Plasmiden vergleichen (Anhang C). Nach der automatischen Präparation der Plasmide wurden zur Qualitätskontrolle Proben hiervon per Agarose-Gelelektrophorese dargestellt. Aus diesen Proben gewonnene DNA-Sequenzen sind als Elektropherogramme im An-

hang C dokumentiert. Die Qualität der Ergebnisse läßt sich mit denen von manuell präparierten Plasmiden vergleichen (Anhang C). Nach der automatischen Präparation der Plasmide wurden zur Qualitätskontrolle Proben hiervon per Agarose-Gelelektrophorese dargestellt. Aus diesen Proben gewonnene DNA-Sequenzen sind als Elektropherogramme im Anhang C dokumentiert.

6 Material und Methoden

6.1 Material

6.1.1 Gerätschaften

Sequenzer:

Perkin Elmer ABI PRISM™ 377

Pharmacia ALF*express*

DNA-Elektrophorese:

BIO-RAD Sub-Cell GT 25cm

Pharmacia ECPS 3000/150

Roboter-Komponenten:

Beckman Biomek 2000

CRS Robotics A465

MJ Research PTC-225

Inkubator:

Adolf-Kühner AG LAB-shaker und LAB-Therm

Zentrifuge:

Jouan CR 322

6.1.2 Chemikalien

AMERSHAM GmbH, Braunschweig:

Thermo Sequenase fluorescent labelled primer cycle sequencing kit with 7-deaza-dGTP, Pimercycle ('Formamidhaltiger Gelbeladungspuffer)

BIORAD GmbH, München:

30% ige Acrylamid-Lösung, Mischungsverhältnis Acrylmid/Bisacryamid beträgt 29:1

BIOZYM GmbH, Olendorf: SeaKem LE Agarose, Hydrolink

DIFCO LABORATORIES, Detroit USA:

Bacto-Yeast-Extract, Bacto-Trytone, Bacto-Agar

PERKIN ELMER GmbH,

Ready Reaction DyeDeoxy Terminator AmpliTaqFS Kit

MERK GmbH, Darmstadt: Borsäure, Natriumchlorid

PHARMACIA GmbH, Freiburg: Tris-Base

QIAGEN GmbH, Hilden:

QIAwell 96 Ultra Plasmid Kit mit den Pufferlösungen:

P1, P2, P3, QW, QE, und PE.

QIAprep 96 Turbo Miniprep Kit mit den Pufferlösungen:

P1, P2, N3, PB und PE.

SERVA GmbH, Heidelberg: APS, TEMED, EDTA

SIGMA GmbH, Deisenhofen: Ampicillin, Ethidiumbromid

USB, Bad Homburg: Harnstoff „ultragrade“ zur Sequenzierung

6.1.3 Puffer und Lösungen

Acrylamid-Gele für ABI 377:

48cm WTR (90 B/h = 2X RUN)

4.25 %iges Acrylamidgel / 7M Harnstoff:

21.0g	Harnstoff
7.1ml	30% Acrylamid
6.0ml	10 * TBE-Puffer
21.0ml	H_2O bidest.

36cm WTR (200 B/h = 4X RUN)

4.5 %iges Acrylamidgel / 6M Harnstoff:

18.0g	Harnstoff
7.5ml	30% Acrylamid
6.0ml	10 * TBE-Puffer
22.0ml	H_2O bidest.

Polymerisation bei beiden Gelsystemen durch Zusatz von 350µl APS (10%) und 15µl TEMED

Agarosegel:

0.8% Agarose in H_2O, 2.5µl Ethidiumbromid Stammlösung pro 100ml wässriger Agaroselösung. Elektrophorese bei 120V.

Ammoniumpersulfat-Lösung 10% (w/v):

1.0 g Ammoniumpersulfat (APS)

Mit H_2O bidest. auf 10 ml auffüllen.

Ethidiumbromid-Lösung: 5 mg/ml in H_2O

Gel-Ladepuffer nach PCR:

Amersham Primercycle (Formamidhaltiger- Fertigpuffer).

Hydrolink-Gele für ALF*express*

Gesamtvolumen	50 ml	60 ml	75 ml
7M Harnstoff	18 g	18.5 g	27 g
1.2 (1.5) xTBE	6 (7.5) ml	7 (9) ml	9 (11.25) ml
6(5)% Hydrolink	6 (5) ml	7 (6) ml	9 (7.5) ml
TEMED	25 µl	30 µl	37.5 µl
10 % APS	250 µl	300 µl	370 µl

Laufpuffer (Hydrolink-Gel) :

Für 1.2 x TBE im Gel: 0.6 x TBE= 120 ml 10 x TBE ad 2l.

Für 1.5 x TBE im Gel: 0.5 x TBE = 100 ml 10 x TBE ad 2l.

10 x TBE-Puffer (1000 ml) für ABI-Gele :

108.0 g Tris-Base.

55.0 g Borsäure.
7.4 g Na_2EDTA.
Mit H_2O auf 1 l auffüllen

10 x TBE-Puffer (1000 ml) für ALF-Gele :

121 .0 g Tris-Base
51.4 g Borsäure
3.7 g EDTA
Mit H_2O auf 1 l auffüllen.

6.1.4 Plasmide

pTZ-18R (PHARMACIA GmbH, Freiburg)

6.1.5 Kulturmedium und Antibiotika

LB-Medium:

5g Bacto-Yeast-Extrakt, 10g Bacto-Trypton, 5g NaCl
mit H_2O dest. auf 1l auffüllen, autoklavieren.
End-Konz. Ampicillin im Medium 250µg/ml.

LB-Agar:

LB-Medium mit 1.5% Bacto-Agar (w/v), autoklavieren.
End-Konz. Ampicillin im Medium 250µg/ml.

Ampicillin: 250mg/ml H_2O bidest. (1:1000)

6.1.6 Bakterienstamm

Escherichia coli, Stamm XL1-Blue (STRATAGENE).

6.1.7 Primer

Die Primer für den ALF*express* unterscheiden sich von den ABI-Primern dadurch, daß sie am 5´-Ende cy5-gelabelled sind.

Universal Primer -40	(5'GTT TTC CCA GTC ACG ACG TTG 3')
Reverse Primer -24	(5'ACA GCT ATG ACA TGA TTA CG 3')
Reverse Primer -48	(5'GCG GAT AAC AAT TTC ACA CAG GA 3')
Universal Primer	(5' Cy5 CgA CgT TgT AAAA CgA Cgg CCA gT 3')
Reverse Primer	(5' Cy5 Cag gAA ACA gCT ATg AC 3')

6.1.8 Verwendte Shotgun-Bank

Für die kommerziell hergestellte Shotgun-Bank wurde der BAC 8F16 von *Arabidopsis thaliana* Chromosom 4 verwendet Die durch Ultraschall-Fragmentierung erhaltenen Fragmente der Größen 1kb und 3kb wurden in den *Sma*I geschnittenen Vektor pTZ-18R kloniert.

6.2 Methoden

6.2.1 Qiagenprotokoll zur Präparation von Plasmid-DNA aus *Escherichia coli*

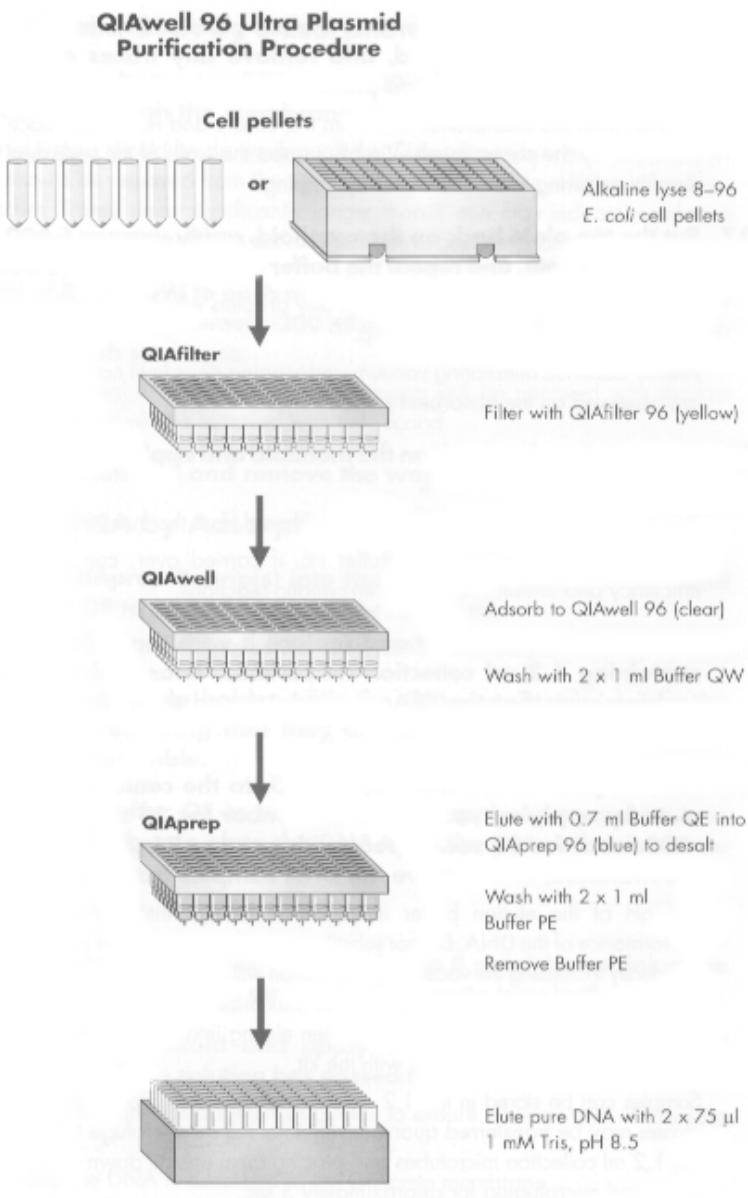


Abbildung 6-1: Qiagen Protokoll zur Plasmid-präparation mit dem Qiawell-Kit (Original aus dem QIAwell System Handbook [Qia95])

Die in dieser Arbeit vorgestellte automatisierte Plasmidpräparation beruht auf dem von Qiagen angebotenen Kit, bei dem die DNA über Mini-Säulen gereinigt wird. Zunächst werden die lebenden Bakterien aus ihrem 96er Raster von Agarplatten mit Hilfe eines entsprechenden Stempels in Anzuchtkammern (96er Mikrotiterplattenformat) übertragen, die jeweils 1.5ml LB-Medium enthalten. Diese Anzuchtkammern werden mit einer selbst-klebenden, sterilen Folie abgedeckt und über Nacht bei 37°C inkubiert (Thermokabinettschüttler).
Nach Abzentrifugieren (1500g für 5 min) wird der Überstand entfernt und die Bakteriensedimente einer alkalischen Lyse unterworfen. Die Methode ist Bestandteil des Qiagen-Kits. In einem Ersten Schritt werden die Zellreste abfiltriert. Anschließend wird die Plasmid-DNA bei niedriger Salzionenkonzentration an eine Anionenaustauschersäule gebunden. Nach erfolgter Bindung wird die Salzionenkonzentration auf einen mittleren Wert erhöht, so daß RNA (0,6-1,2 M NaCl), Proteine und niedermolekulare Verbindungen (0,05-0,2 M NaCl) entfernt werden. Die Elution der Plasmid-DNA erfolgt über einen Hochsalzpuffer (1,4-1,6 M NaCl). Abschließend erfolgt eine Entsalzung durch Präzipitation mit Isopropanol.

6.2.2 Vorbereitung der Glasplatten

Die Glasplatten werden vor dem Lauf mit einem Detergenz (Alconox) behandelt und mit destilliertem Wasser gespült. Mit einem (9:1) Gemisch aus Isopropanol und H_2O werden die Platten nachbehandelt. Sind die Glasflächen trocken, dann werden die Spacer aufgesetzt und die Gelkassette zusammengebaut.

6.2.3 Gel gießen

Der Harnstoff wird nach dem Ansatz der Gellösung gelöst, wobei nicht über 50°C erwärmt werden soll. Die Lösung wird über ein 0.2 µm Filter geklärt. Das Filtrat muß vor dem Einsatz entgast sein. TEMED und 10% ige APS-Lösung werden in den entsprechenden Mengen zum Schluß zugegeben, wodurch die Polymerisation startet. Die Flüssigkeit wird nun zügig zwischen die Glasplatten der Gelkassette gegossen, um dort für die nächsten 2 Stunden zum Gel zu polymerisieren. Nach Einfüllen der Polymerisationslosung wird sogleich der Kamm eingesetzt, damit die Auftragungstaschen entstehen können. Während der Polymerisation sollte dieKassette horizontal lagern.

6.2.4 Sequenzierung mit dem ABI 377

6.2.4.1 Manuelle Sequenzierung

Ansatz für PCR-Sequenzierung (ABI-Verfahren):
2 µl PreMix (R6G+ROX+R110+TAMRA) Didesoxinucleotide. 5 pmol (bzw. 10 pmol) Primer (UPILC) 1µl 0.5 µl DMSO (= 5%) zur Stabilisierung der Einzelstränge. ca. 2 µl DNA (0.4-0.6 µg/µl) je nach photometrischer Bestimmung bei 320nm. Endvolumen soll 10µl sein. Dies wird durch H_2O aufgefüllt.

Cyceln: 96°C - 3 min
30 Zyklen: **96°C - 10 s**
60°C - 4 min, Lagern bei 4°C.

DNA Präzipitation:
10µl H_2O zugeben.
10 µl 2mmolar $MgCl_2$
45 µl 95% Ethanol.

Alles mischen und 15 Minuten bei Zimmertemperatur stehen lassen. Dann Abzentrifugieren bei 2000g für 25 Minuten. Das Präzipitat ist als Pellet zu sehen. Anschließend Mikrotiterplatte umkehren und restliche Flüssigkeit bei 100g für 1 Minute ausschleudern. Anschließend für 3 Minuten bei 55°C trocknen.

Auftragung der Proben auf das Sequenziergel:
Die Proben werden in 3µl Primercycle-Puffer aufgenommen. Vor dem Auftrag auf das Gel werden die Proben für 3min auf 95°C erhitzt. Es werden 1.5µl Probe aufgetragen.

6.2.4.2 Strategie für den Pipettierroboter

Pipettierschritte

8µl werden mit einer 8 fach Multipipette aus der MP I, Reihe 1 auf alle 96 Positionen der PCR-Reaktionsmikrotiterplatte verteilt. Anschließend werden jeweils 2µl DNA-Proben aus der MP II zugegeben.

Ansatz für PCR-Sequenzierung (ABI-Verfahren):

PreMix= 192µl
Primer= 96µl
DMSO= 48µl
H_2O= 432µl

Dies ergibt zusammen den „Ready Mix“ zu 768µl, der dann in 8 Portionen zu 96µl in die erste Reihe einer Mikrotiterplatte (MP I) vorgegeben wird.

Cyceln:

96°C - 3 min
30 Zyklen: **96°C - 10 s**
60°C - 4 min
Lagern bei 4°C.

DNA Präzipitation:

960µl H_2O + 960µl $MgCl_2$ Lösung werden als 8 x 240µl in die zweite Reihe der MP I vorgelegt. In die dritte Reihe der MP I kommt 96% Ethanol zu 8 x 540µl. Um eine Verdunstung des Ethanols zu vermeiden wird die gesamte Mikrotiterplatte mit einem Deckel abgedeckt. Vor der eigentlichen Pipettierung nimmt der Hilfsroboter den Deckel ab. Formamidpuffer wird in der vierten Reihe der MP I zu 8 x 36 µl vorgelegt. In eine zweite Mikrotiterplatte (MP II) werden 96x 2µl DNA-Proben vorgelegt.

6.2.4.3 Elektophoresebedingungen ABI 377-Sequencer:

Für das Gelsystem 48cm WTR (90 B/h = 2X RUN) 3000V, 50mA, 48W, 10h, 51°C Geltemperatur, 40mW Laserleistung. Für das Gelsystem 36cm WTR (200 B/h = 4X RUN) 2700V, 60mA, 200W, 4h, 51°C Geltemperatur, 40mW Laserleistung. Für beide Gelsystem wird als Laufpuffer 1 * TBE verwendet.

6.2.5 Sequenzierung mit dem ALF*express*

6.2.5.1 Manuelle Sequenzierung

PCR-Sequenzierung (ALF-Verfahren)
Eine Probe muß auf 4 Reaktionen (AGTC) verteilt werden. Es werden zunächst 2µl - 2.5µl Nukleotide vorgelegt (AGTC). Reaktionsgemisch:

1µg/µl DNA = 4-5µl Probenlösung.
Primer 3µl
Mit H_2O auf 24µl auffüllen.
Von diesem Reaktionsgemisch werden 6µl auf die A, G, T und C Vorgaben verteilt.

Cyceln:

Es wird das Programm „EMBL“ in der „PTC-225“ (M.J.-Research) aufgerufen:

	95°C - 3 min
25 Zyklen:	95°C - 15 s
	55 °C - 30 s
	68 °C - 30 s

Aufbewahren bei 4°C.

Auftragung der Proben auf das Sequenziergel:
5µl Stoplösung hinzufügen. Auftragungsvolumen auf das Sequenziergel: 3µl

6.2.5.2 Elektrophoresebedingungen für den ALF*express*

Für 0.5 mm Hydrolink-Gele: 1500 V, 70 mA, 15W nach 15min auf 30W erhöhen, 45°C, 80min. Sampling-Rate: $2s^{-1}$ Für 0.3 mm Hydrolink-Gele: 1500V, 40mA, 15W nach 15min auf 30W erhöhen, 45°C, 480min. Sampling-Rate: $2s^{-1}$ Für beide Gelsysteme wird als Laufpuffer 0.5 * TBE verwendet.

7 Diskussion

Die beschriebene Anlage besteht aus industriell gefertigten Labormaschinen und Eigenentwicklungen. Während der Planungsphase wurde festgelegt, daß dem Laborpersonal alle funktionalen Elemente dieser Anlage auch für manuelle Arbeiten zur Verfügung stehen. Somit kann auf einfache Art und Weise von einer vollautomatischen Routineprozessierung auf interaktive und somit individuelle Labortätigkeit gewechselt werden. Die Integration von industriell angebotenen Insellösungen (Biomek 2000, PTC 225) in eine kompakte Arbeitsstation zur Prozessierung arbeitsintensiver Labor-Routinemethoden ist in der vorgestellten Anlage gelungen. Es entstand eine nicht rückgekoppelte Anlage mit begrenzter Kapazität (6 Plasmid-Präparationen mit nachgeschalteter PCR-Sequenzierung in ca. 6 Stunden, vgl. Kapitel 7.2). In noch auszuführenden Schritten wird der Anlage schrittweise weitere Funktionalität zugefügt, um zum einen die Kapazität auf eine 24 stündige Auslastbarkeit zu steigern (vgl. Kapitel 7.2) und zum andern ein Qualitätssicherungssystem (vgl. Kapitel 7.3) zu integrieren.

7.1 Bei nicht rückgekoppelten Robotersystemen verbleibt ein Restrisiko während unüberwachter Produktion

Als „Risiko“ soll dabei der Arbeitsausfall des Systems mit mechanischem Folgeschaden bezeichnet werden.

Werden zur Präparation notwendige Elemente transportiert, so werden die bewegten Gegenstände nur ausnahmsweise ideal an den Zielort eingestellt. In den meisten Fällen werden die einzustellenden Objekte an den Zielpositionen mit geringfügigen Lageänderungen präsentiert. Das zu bewegende Element wird dabei während des Transportes um so mehr aus seiner idealen Lage bewegt, je größer die Anzahl beeiflussender Faktoren in die gleiche Richtung weisen. Sind beeinflussende Faktoren exakt entgegengerichtet, so kann dies in seltenen Fällen eine Kompensation bewirken. Mögliche Fehlerquellen sind dabei:

- Toleranzen in der Aufnahmevorrichtung an der lagernden Position. Befindet sich das zu transportierende Objekt beispielsweise im Karussell, so sind die Toleranzen dort durch folgende Punkte definiert:

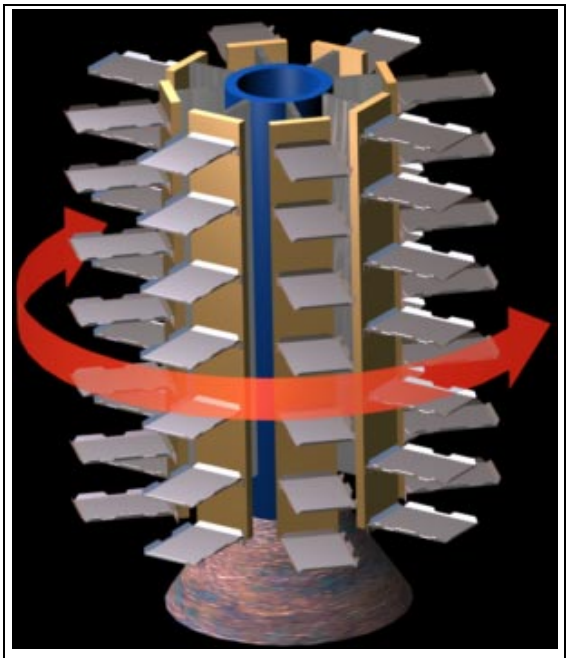


Abbildung 7-3: Positionierungsungenauig-keiten aufgrund Motorstellfehler.

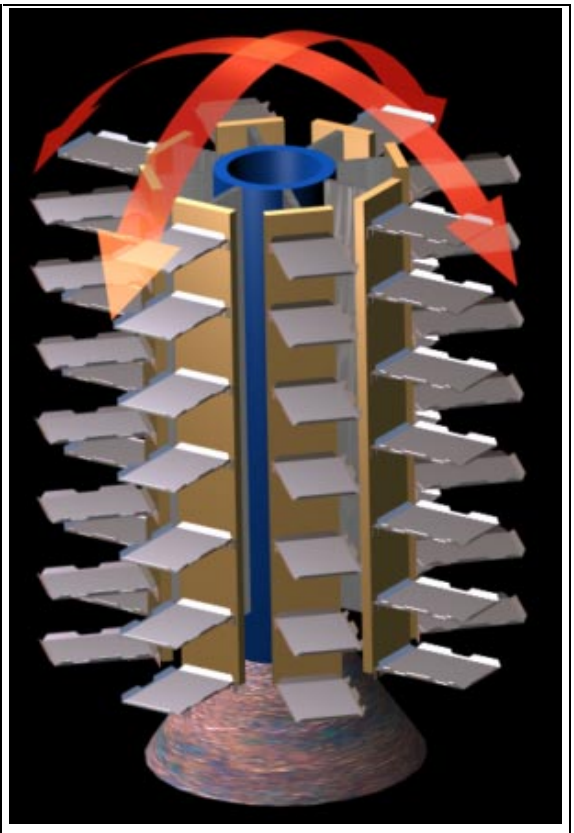


Abbildung 7-2: Abweichung durch Auslenkungen der vertikalen Hauptachse.

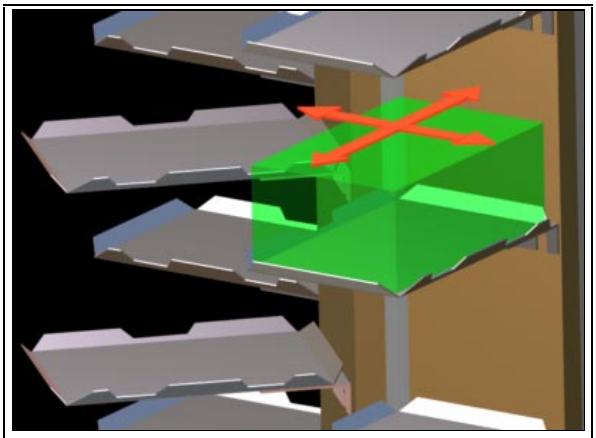


Abbildung 7-1: Positionierungsfehler aufgrund fertigungsbedingt unterschiedlicher Grundflächen der einzustellenden Objekte.

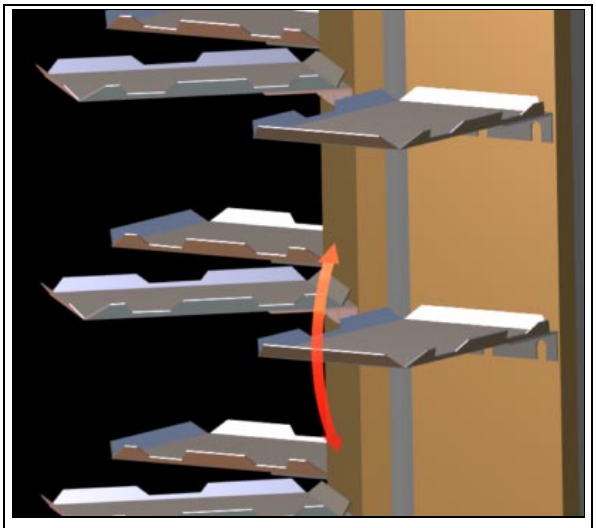


Abbildung 7-4: Positionierungsfehler aufgrund unterschiedlicher Materialbeanspruchung der aufnehmenden Einstellschalen.

1. Positionierungsungenauigkeiten des an-treibenden Motores im Fuß des Karussells (Abweichungen im horizontalen Präsentationswinkel, vgl. Abbildung 7-3).

2. Während der Rotation des Karussels kann eine geringfügige Abweichung der vertikalen Hauptachse durch Präzession beobachtet werden. Dies führt zu einer

geringfügigen Abweichung in der Inklination und Deklination der präsentierten Objekte (vgl. Abbildung 7-2).
3. Das zu transportierende Objekt liegt mit seiner Grundfläche in der Ablageschale des Karussells. Diese Grundfläche weist Fertigungstoleranzen auf, wodurch sich Lageabweichungen horizontal innerhalb der Ablageschale ergeben (vgl. Abbildung 7-1).
4. Die Ablageschale besteht aus zwei miteinander verlöteten Blechen, wobei über das vertikal angeordnete kurze Blech die Befestigung an der Hauptachse des Karussells bewerkstelligt ist. Über Fertigungstoleranzen und Ermüdungserscheinungen des Materials an der Lötstelle ergeben sich geringfügig unterschiedliche Neigungswinkel für die vertikal im Karussell eingestellten Objekte (vgl. Abbildung 7-4).

- Toleranzen während des Verfahrweges über den CRS 465. Der Greifer faßt das zu transportierende Objekt mit gummibemantelten „Fingern“ an. Ist die Oberfläche des zu greifenden Gegenstandes sehr glatt oder eingefettet (Handcrème), so kann dieser aufgrund der auf ihn einwirkenden Beschleunigungskräfte während des Transportweges eine Lageänderung erfahren. Weil diese Bewegung des gegriffenen Objektes durch zahlreiche Faktoren beeinflußt ist, die nicht unmittelbar bestimmt werden können, muß der resultierende relative Lagevektor $\wp=(x,y,z)$ als klein, aber zufällig beschrieben werden.
- Unsicherheiten in der globalen Anordnung aller größeren Systeme. Die größeren Geräteplattformen (Biomek 2000, PTC 225, Karussell, Seitenregal, Aufnahmeblock der Vakuumkammer, Tischsegmente) können als Ebenen aufgefaßt werden. Durch Fertigungstoleranzen stellen diese jedoch häufig Torsionsflächen dar. Die in dieser Arbeit beschriebene Anlage befindet sich auf einem Tisch, mit drei nicht ausgerichteten Plattensegmenten. Genaugenommen gibt es daher drei Bezugssysteme, wobei sich einige Geräteelemente aufgrund ihrer Dimension in zwei dieser Bezugssysteme befinden (Aufnahmeblock der Vakuumkammer und Seitenregal). Relativ zu den drei Grundplatten existieren jeweils Bezugssystem-Vektoren. Diese sind aufgrund einer generell vorhandenen Torsion der jeweiligen Grundplatten (Kunststoff) ortsabhängig und daher sehr komplex. Deren Einfluß wird dadurch gemindert, daß ausschließlich der CRS 465 Roboter Objekte über die Grenzen der verschiedenen Stationen verschiebt. Somit vereinfacht sich die Situation dahingehend, daß der CRS 465 sein Bezugssystem über das gesamte System „spannt“. Die ansonsten notwendigen Translationsvektoren für die drei Plattensegmente entfallen dadurch. Lagevektoren der Geräteplattformen relativ zum Bezugssystem des CRS 465 gehen mit dieser Betrachtungsweise in die „Toleranzen“ der entsprechenden mechanischen Aufnahmen ein. Ein Ausgleich erfolgt über die manuell zu bestimmenden Einstellwinkel des CRS 465 („teachen“).

Zieht man all diese Faktoren in Betracht, so ist die Notwendigkeit mechanisch korrigierender Hilfseinrichtungen offensichtlich. Die in dieser Arbeit vorgestellte Anlage verfügt über zahlreiche derartiger Sicherheitseinrichtungen, die in ihrer Toleranzkompensation eine ausreichend sichere Arbeit des Gesamtsystems gewährleisten. Dennoch bleibt bei einem nicht rückgekoppelten System dann ein Restrisiko bestehen, wenn diese Sicherheitseinrichtungen versagen. Dies tritt immer dann ein, wenn sich das Objekt außerhalb der Toleranzkompensation befindet. Um das Risiko zu minimieren kann man die Toleranzkompensation des Systems erweitern. Diese Strategie wird allerdings rasch an Grenzen stoßen und stellt daher keine endgültig befriedigende Lösung des Problems dar. Zum einen wird der mechanische Aufwand für entsprechend zu konstruierende Führungen zu groß, zum andern übersteigen die Herstellungskosten einer solchen Anlage ihren ökonomischen Nutzen. Eine andere Strategie zur Minimierung des Betriebsrisikos verfolgt die ständige Kontrolle des Arbeitsablaufes einerseits und möglicher Korrekturen in Problemsituationen andererseits. Ausgehend von einer ausreichend stabil arbeitenden nicht-rückgekoppelten Anlage kann diese Strategie als „add-on“ Paket verwirklicht werden. Die schon bestehende Sicherheit wird durch sensorische Rückkoppelung kontrolliert und gegebenenfalls erweitert. Ein weiterer Vorteil dieser Vorgehensweise besteht darin, die Anlage schrittweise in die völlige sensorische Rückkoppelung zu überführen. Das objektorientierte Softwaredesign ermöglicht die Ableitung eines rückgekoppelten virtuellen Roboters aus dem in dieser Arbeit vorliegenden virtuellen Roboter. Als Basisklasse kann in diesem Zusammenhang der Controller dienen. Eventuell ist noch die Controller-Schnittstellenklasse um den einen oder anderen Ereigniskanal zu erweitern.

7.1.1 Rückkoppelungsstrategien

Um eine nicht rückgekoppelte Anlage mit entsprechender Sensorik auszustatten können unterschiedliche Strategien verfolgt, bzw. sich wechselseitig optimierend miteinander gekoppelt werden. Prinzipiell kann eine Anlage entweder über viele Sensoren, die sich an allen kritischen Stellen befinden, oder aber über eine dynamische, situationsbedingte Analyse überwacht werden.

Eine Kombination beider Strategien kann die Vorteile der jeweiligen Methode nutzen, um die Nachteile der anderen zu kompensieren. Dies setzt jedoch einen entsprechend sorgfältig ausgearbeiteten Entwurf der übergeordneten Software voraus. Um Strategien in der erwähnten Art und Weise kombiniert einzusetzen, bedarf es eines geeigneten Strategiemusters und einer gewählten Hierarchie der einzusetzenden Methoden. Weil diese von der zu kontrollierenden Situation bestimmt wird, sollte das entsprechende Objekt in einer eigenen Klasse gekapselt werden. Ein geeigneter Name wäre dann z.B. „SensorManager“, der von seiner Natur her am besten über eine Relation in der „Welt“ des betreffenden Roboters untergebracht wäre. Dort könnte sich dieser Manager am besten an den vorhan-

denen Daten orientieren und die eine oder andere Strategie aus dem lokal vorhandenen Sensorarsenal auswählen.

7.1.1.1 Multisensorische Rückkoppelung

Viele Sensoren sind an den kritischen Stellen der Anlage verteilt. Der Vorteil dieser Strategie liegt in ihrer potentiell massiv parallel auswertbaren Struktur. Jeder Sensor kann so geschaltet sein, daß eine „Ja/Nein“ Antwort auf die Frage „korrekte Situation ?“ gegeben wird. Einsetzbar wären solche einfachen Überwachungsstrukturen beispielsweise in den Aufnahmen des Biomek 2000, des Seitenregalbereichs und des Ventilationssystems. Wird ein Objekt über den CRS 465 von der Arbeitsplattform entfernt, kann eine Anfrage an entsprechend sensitive Einheiten (z.B. Lichtschranken) in der entsprechenden Aufnahme kontrollieren, ob sich das Objekt noch im Bereich der Aufnahme befindet oder entfernt wurde. Der Arbeitsschritt kann aber erst dann als erfolgreich abgeschlossen gelten, wenn sich das Objekt am Zielpunkt korrekt plaziert einfindet. Auf diese Weise kann kontrolliert werden, ob das Objekt auf seinem Weg zum Zielort verlorengegangen ist, oder nicht. Wichtige Kontrollpunkte sind neben den Aufnahmen am Biomek 2000 das Seitenregal und die Überwachung der korrekten Beschickung der Vakuumkammer. Die Dichtigkeit der Kammer wird am besten über Drucksensoren, die in die Seitenwände des Unterteils eingearbeitet werden kontrolliert. Die Daten dieser Sensoren dokumentieren die Unterdruckverläufe im unteren bzw. mittleren Kammerbereich. Die angegebene Sensorik kann die Frage der korrekten Orientierung einer Objektplazierung nur unter erheblichem Aufwand beantworten, was ein Nachteil dieser Strategie ist. Besonders deutlich wird dieser Nachteil für die automatische Überprüfung richtig eingestellter Filterträger. Probenmuster können dadurch verlorengehen, daß Filterträger versehentlich und unbeobachtet um 180° übereinander in die Vakuumkammer gestellt werden. Damit dies nicht geschehen kann, ist die Vakuumkammer so konstruiert, daß die Filterträger nur in einer definierten Richtung eingestellt werden können. Somit ist zwingend gewährleistet, daß ab Entnahme der Bakterienlysate aus den Anzuchtgefäßen kein Vertauschen des Pipettierrasters erfolgen kann. Folglich ist es zwingend notwendig, die Filterträger vor der Präparation in bestimmter Ausrichtung in das Karussell einzustellen. Erfolgt hier ein Fehler, befindet sich der Filterträger sofort weit außerhalb der Toleranzkompensation des Systems und ein mechanischer Crash ist die Folge. Eine Überprüfung der korrekten Orientierung dieser Filterträger ist über eine einfache Sensorik, insbesondere im Karussell schwer zu verwirklichen. Dieser Nachteil kann über die visuelle Rückkoppelung ausgeglichen werden.

7.1.1.2 Visuelle Rückkoppelung

Eine Kamera, welche sich z.B. am Greifer des CRS 465 befindet, nimmt ein Bild der aktuellen Szene auf. Das gewonnene Bild wird in einer digitalen Bildanalyse untersucht und liefert die gewünschte Information. Der Vorteil dieser Strategie

liegt in Aussagen über die gerade vorliegende Bildszene. So können beispielsweise im Karussell fehlorientierte Filterträger anhand ihrer charakteristischen Kantenstruktur identifiziert werden. Der Karussellbereich ist aufgrund seiner Variabilität für multisensorische Rückkoppelung schlecht zugänglich. Über eine visuelle Rückkoppelung ist beispielsweise auch die Detektion der korrekten Anzahl Pipettenspitzenkästen, Elutionsauffänger und Mikrotiterplatten nebst PCR-Einsätzen möglich. Die dem System übergeordnete Software kann sich somit zu Beginn einer Präparation einen Überblick verschaffen, ob und wo Beschickungsfehler vorliegen und im gegebenen Fall darauf hinweisen. Über eine Erweiterung der integrierten Datenbank kann auch die exakte Plazierung von Labware innerhalb des Karussells unnötig werden. Das visuelle System zählt die notwendigen Objekte durch und stellt deren Position im Karussell anhand des aktuellen Drehwinkels relativ zur ersten Reihe und Position innerhalb dieser Reihe fest. In der internen Datenbank sind die entsprechenden Daten festgehalten und stehen bei Bedarf während der unterschiedlichen Präparationen zur Verfügung. Die visuelle Rückkoppelung birgt zwei Nachteile in sich. Diese Nachteile sind so entscheidend, daß ein genereller Einsatz dieser Methode unvorteilhaft ist. Zum einen ist der Zeitaufwand für eine eindeutige Interpretation einer Szene sehr groß. Zum anderen ist das Ergebnis einer digitalen Bildanalyse nicht immer eindeutig und bedarf der Nachuntersuchung. Die gegenüber einer multisensorischen Rückkoppelung erheblich längere Reaktionsdauer begründet sich auf dem unter Umständen immensen Rechenaufwand bei der Analyse komplexer Szenen. Um beispielsweise die korrekte Position der beiden Filterträger einer Qiagen Turbo Plasmid-Präparation festzustellen bedarf es neben einer eindeutigen Kantendetektion (stumpfes Ende der Filterträger) auch der Farbdetektion. Beide Filter unterscheiden sich lediglich durch das Vorhandensein oder Fehlen einer Farbe (farbloses Filter= Lysefilter, blaues Filter= Elutionsfilter). Somit ist die Notwendigkeit einer farbbasierten Bildanalyse gegeben. Um normalisierte Farbwerte zu erhalten ist desweiteren der Einsatz einer geeigneten Beleuchtung (am besten an der beobachtenden Kamera) und die Kontrolle ihrer Funktionsfähigkeit erforderlich. Zur Beschleunigung der Szeneninterpretation kann die Kantendetektion über ein Graustufenbild erfolgen, die eindeutige Filterträgerzuordnung hingegen durch integrative Farbbestimmung in einem eindeutigen Zielbereich geeigneter optischer Dichte innerhalb der detektierten Struktur. Die Kantendetektion kann dadurch erschwert werden, daß sich die Kamera am Greifer des CRS 465 im dreidimensionalen Raum bewegt. Je nach Positionierung (Auskunft über die Stellwerte der 7 Achsen des CRS 465) sollte eine Transformationsmatrix zur Normierung der Szene aufgebaut werden. Diese Erschwernis kann jedoch auch ein Vorteil in kritischen Situationen sein. Kann unter bestimmten Bedingungen kein Ergebnis der Bildanalyse erhalten werden, so kann der CRS 465 die Perspektive der Szene über Veränderung des Blickwinkels geringfügig verändern. Ein Vergleich beider Bilder kann sogar eine Abstandsmessung oder Objektbestimmung ermöglichen. Dies könnte dann notwendig werden, wenn ein Objekt identifiziert werden soll,

das vom Greifer verloren wurde. Das System sollte zu Beginn einer Reihenpräparation alle visuell zu kontrollierenden Positionen „anfahren“ und deren korrekte Lage bestimmen. Im Fehlerfall sollte ein Dialog erfolgen. Nach erfolgreicher Überprüfung startet das rückgekoppelte System die geplanten Präparationen. Das visuelle Erkennungssystem wird nur noch dann aktiviert, wenn sich aus dem multisensorischen Kontrollbereich widersprüchliche Daten ergeben. Für einen 24 stündigen Betrieb der Anlage kann die visuelle Rückkoppelung zur Optimierung der Logistik herangezogen werden. Man könnte Verbrauchsmaterial nach Ablauf der 24 Stunden komplett ergänzen, was als Strategie jedoch relativ starr ist und eine Pausierung des Systems verursachen würde. Eine elegantere Lösung bietet die visuelle Rückkoppelung, wenn während der Präparation der Bestand der Verbrauchsmaterialien kontrolliert, willkürliche Ergänzungen registriert und die entsprechende Datenbank informiert wird. Das System könnte seinen Bedarf kontinuierlich anzeigen, wobei das Verbrauchsmaterial von außen an prinzipiell beliebiger Stelle im Regalbereich eingestellt werden kann. Die visuelle Rückkoppelung stellt die entsprechende Position fest und legt die zugehörigen Daten ab. Der Bedarf wird dann aus dem sich so ergebenden Pool gedeckt.

7.2 Steigerung des Probendurchsatzes für die vorgestellte Anlage

Der theoretisch mögliche Probendurchsatz der in dieser Arbeit vorgestellten Anlage beträgt in 6 Stunden 576 Plasmidpräparationen. Wird der Plasmidpräparation eine PCR-Sequenzierung im ABI-Schema nachgeschaltet, so laufen die betreffenden Reaktion in einer Mikrotiterplatte im 96er Format ab. Diese Mikrotiterplatte nimmt eine weitere Position im Karussellbereich ein. Werden 576 Plasmidpräparationen zur PCR-Sequenzierung eingesetzt, so ist der sich daraus ergebende Bedarf an PCR Mastermix: 16µl x 576 Plasmidpräparationen = 9.216 ml. Dieses Volumen kann nur in einem weiteren Gefäß bereitgestellt werden. Daher ist für eine der Plasmidpräparation nachgeschaltete PCR-Sequenzierung eine weitere Position für das entsprechende Reservoir zu reservieren, das allerdings für alle 576 Präparationen hinreichend ist. Das mit „Mastermix“ befüllte Gefäß kann daher auf eine der vom CRS 465 nicht erreichbaren Positionen des Biomek 2000 abgestellt werden (z.B. Platz A/B 1). Für die zur Plasmidpräparation notwendigen Verbrauchsmaterialien bietet die Anlage folgende Kapazität:

24 Positionen im Regalbereich.
56 Positionen im Karussellbereich (8 Reihen à 7 Schalen) mit großem vertikalen Abstand für Pipettenspitzenhalter und Filterträger (die Höhe dieser Verbrauchsmaterialien macht große vertikale Abstände zwischen den Schalenhaltern erforderlich).
= 80 mögliche Positionen.

Pro Präparation werden im Karussellbereich eingestellt:
2 Filterträger, 1 Eluatfänger, 6 Pipettenspitzenkästen.
Für eine Präparation werden somit 9 Stellplätze im Karussell benötigt. Das Karussell weist mit 56 verfügbaren Positionen eine Kapazität auf, die für 6 nacheinander ausgeführte Präparationen ausreicht. Pro Präparation werden im Regalbereich eingestellt:
1 Gefäß für 96 Bakterienkulturen, 2 Pufferreservoirs (+ 1 Mikrotiterplatte für PCR). Somit werden im Regalbereich 3 (+PCR: 4) Positionen pro Präparation benötigt. Das Regalsystem bietet somit eine Kapazität, die für 8 (+PCR: 6) nacheinander ausgeführten Präparationen ausreicht. Die Verbrauchsmittel müssen in der gegenwärtigen Entwicklungsphase nach ca. 6 Stunden Systemtätigkeit erneuert werden. Um die Anzahl der verwendbaren Stellplätze für Labormaterial zu erhöhen ist ein besonderes Regalsystem zu konstruieren. Der CRS 465 Roboter, welcher den Transport des Verbrauchsmaterials übernimmt, hat eine begrenzte Reichweite. Diese ist sowohl vertikal als auch in seinem horizontalen Arbeitskreis durch den Radius seines Armes begrenzt. Geometrisch läßt sich die Grenze seiner Reichweite am besten als Innenfläche eines Torsionsellipsoids beschreiben. Das zu konstruierende Regal weist somit am besten eine Krümmung auf, die dem Reichweitenradius des Greifarms entspricht. Darüberhinaus soll dieses Regal mehrere Ebenen aufweisen. Jede Ebene kann durch eine Steuerungselektronik in die Reichweitenebene des Greifarms gehoben (bzw. gesenkt) werden. Um die Anlage 24 stündig zu betreiben ist das Regal wie folgt zu konzipieren:
Pro Plasmid-Präparation mit anschließender PCR-Sequenzierung im ABI-Verfahren sind notwendig:
1 Gefäß für 96 Bakterienkulturen, 2 Filterträger, 1 Eluatfänger, 6 Pipettenspitzenkästen, 1 Mikrotiterplatte (PCR-Reaktion)= 11 Elemente für 1 Plasmid-Präparation mit nachgeschalteter PCR. Durch zeitversetztes Starten der PCR-Reaktion kann in einer seriellen Abarbeitung jede Stunde eine komplette PCR-Sequenzierung aus der PTC 225 entnommen werden (vgl. Abbildung 7-5). Somit sind theoretisch stündliche Plasmid-Präparationen mit nachfolgender PCR-Sequenzierung möglich. Somit sind für 24 Stunden insgesamt 24x11=264 Labormittel im zu konstruierenden Regalsystems zu präsentieren. Vorstellbar sind 66 Elemente in 4 Ebenen bzw. 44 Elemente in 6 Ebenen. Die Ebenen werden je nach augenblicklichem Bedarf motorisch in die Reichweite des Roboters gefahren. Dies ist technisch über eine entsprechende Positionsrückkoppelung mit ausreichender Präzision realisierbar. Eine Schrittmotorsteuerung, erzeugt über ein Präzisionsgetriebe eine entsprechende Kraft auf eine Schneckengangwendel, die das Regalsystem auf die notwendige Ebene hebt. Lichtschranken mit entsprechender Optik können als Basis für ein geeignetes Rückkoppelungssystem dienen. Die Regalebene kann auf diese Weise reproduzierbar in den Reichweitenradius des CRS 465 gebracht werden. Die Softwareansteuerung kann über eine Schnittstelle des in dieser Arbeit vorgestellten „Virtuellen Roboters“ erfolgen.

Somit sind über die auf diese Weise erweiterte Anlage in 24h insgesamt 2304 Plasmidpräparationen mit nachgeschalteter PCR Sequenzierreaktion möglich. Die zur Präparation notwendigen Puffer- und Reaktionslösungen werden aus einem Bereich mit automatischer Wiederbefüllung zur Verfügung gestellt und nicht mehr im Regalsystem untergebracht (vgl. Kapitel 7.2.1).

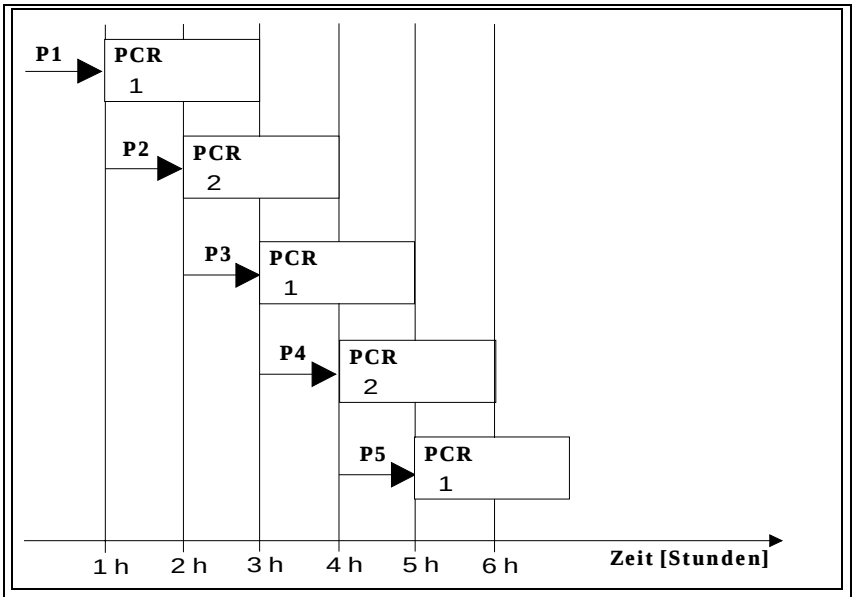


Abbildung 7-5: Zeitversetztes Starten der PCR-Reaktion in der PTC 225. Die Plasmid-Präparationen (Pn; n=1 .. 5) werden seriell über die automatische Prozessierung des Qiagenprotokolls auf der beschriebenen Anlage nachgeliefert.

7.2.1 Automatisierte Wiederbefüllung von Pufferreservoiren

Die während der Präparationen eingesetzten Puffer und Reaktionslösungen weisen stets gleiche Zusammensetzung auf. Somit ist es unnötig, an vielen Positionen innerhalb des Seitenregals die gleichen Puffer in Reservoirs bereitzustellen. Es genügen hierfür lediglich ein festgelegter Bereich im System, bei dem die entsprechenden Reservoirs nach Entleerung automatisch aus entsprechenden Tanks wiederbefüllt werden. Dieser Bereich muß nicht im Regalsystem lokalisiert sein, sondern kann an beliebiger Stelle im System eingerichtet werden, sofern diese vom CRS 465 erreicht werden kann. Der Flüssigkeitsspiegel in den zu befüllenden Reservoiren kann durch Ultraschall oder mittels Laserstrahl bestimmt werden. Die für die Befüllung verantwortlichen Ventile können über die eingehenden Meßdaten geregelt werden. Das Abmessen und Bereitstellen von Puffern in entsprechenden Reservoirs ist arbeits- und personalintensiv und daher entsprechend teuer. Durch eine automatisierte Wiederbefüllung ist ein ökonomischerer Betrieb der Anlage möglich. Ein weiterer Vorteil besteht darin, daß die Puffer nicht über längere Zeit „offen“ stehen, was bei dem ethanolhaltigen Waschpuffer besonders ungünstig ist.

7.2.2 Erweiterte Abfallentsorgung

Durch die wiederholte Plasmidpräparation über die vorgestellte Anlage ist es notwendig, das sich ergebende Abfallvolumen über die Vakuumfalle abzuleiten. Ein entsprechend chemikalienresistentes Ventil wird zu diesem Zweck am Boden der Vakuumfalle angeschlossen. Nach Abschluß einer Plasmidpräparation wird Ventil 3 (vgl. Abbildung 5.12) geöffnet (Belüftung der Falle) und das Abfallvolumen durch dieses chemikalienresistente Ventil aus dem System entfernt.

7.3 Qualitätskontrolle präparierter DNA

Für eine automatisch arbeitende Anlage mit Massenproduktion ist es notwendig die aktuellen Präparationsschritte zu dokumentieren. Dieser Aspekt ist insbesondere für eine industrielle Nutzung der Anlage von Bedeutung, weil hierdurch erst der Qualitätsstandard des Produktes nachgewiesen werden kann. Fehlerereignisse, Protokolle von Systemdaten, wie z.B. Unterdruckverlauf, Temperatur und Inkubationsdauer während des ausgeführten Protokolls, sowie Ergebnisse der Qualitätsmessungen sollen in einer entsprechenden Datei dokumentiert werden. Eine besondere Möglichkeit, einen möglichst ökonomischen Probendurchsatz zu erhalten ist, die Qualität der gewonnenen Plasmide vor Einsatz in die PCR-Sequenzierung zu testen. Schlecht präparierte DNA sollte durch solche entsprechend guter Qualität aus anderen Präparationen ersetzt werden. Diese Manipulationen machen allerdings ein ausgefeiltes Datenbanksystem erforderlich, um nicht den Überblick über die einzelnen Proben zu verlieren. Zur automatischen Aktualisierung des erforderlichen Datenbanksystems bietet sich ein Barcode Schreib- und Lesesystem an, das sich als eigene Einrichtung an einer definierten Stelle im Gesamtsystem befinden kann. Die datenbankrelevanten Gefäße können z.B. über einen CRS 465 an die betreffende Stelle der Anlage gebracht und dort markiert bzw. überprüft werden.

7.3.1 UV-Messung

Die DNA-Konzentration kann mit der Ultraviolettabsorptionsspektrometrie sehr genau gemessen werden. Eine DNA-Lösung absorbiert Licht von 260 nm Wellenlänge. Die Menge der absorbierten ultravioletten Strahlung ist direkt proportional zur DNA-Konzentration. Ist der Absorptionswert 1.0, so entspricht dies einer Konzentration von 50 µg doppelsträngiger DNA in einem Mililiter Lösung. Über die Ultraviolett-Messung kann auch die Reinheit einer DNA-Präparation bestimmt werden. Verunreinigungen mit Protein oder Phenol können durch Absorptionsmessungen bei 280 nm erfaßt werden. Der Quotient aus den Absorptionsdaten von 260 nm und 280 nm kann als direktes Maß für die Qualität der gewonnenen doppelsträngigen DNA dienen. Eine reine DNA Probe weist einen Quotient von A_{260}/A_{280}= 1.8 auf. Ist der Anteil an den oben erwähnten Verunreinigungen gößer, so ergeben sich geringere Werte für den Quotienten.

7.3.2 Fluoreszenzmessung

Doppelsträngige DNA kann angefärbt werden. Man nutzt dabei die Tatsache aus, daß sich bestimmte Farbstoffe in die Doppelhelix der DNA einlagern („interkalierende Farbstoffe“). Wird eine auf diese Weise angefärbte DNA-Probe mit ultraviolettem Licht bestrahlt, so werden die Farbstoffmoleküle zur Fluoreszenz angeregt. Die Intensität des emittierten Lichtes wird dann gemessen. Neben Ethidiumbromid bietet sich auch SYBR Green I als geeigneter interkalierender Farbstoff an. Der Vorteil von SYBR Green I gegenüber Ethidiumbromid liegt darin, daß mit diesem Farbstoff additiv gefärbt werden kann. Eine Überfärbung des Präparates tritt nicht ein. Eine Differenzierung der Färbung ist also nicht notwendig, was den Einsatz dieses Farbstoffes zur quantitativen Bestimmung des

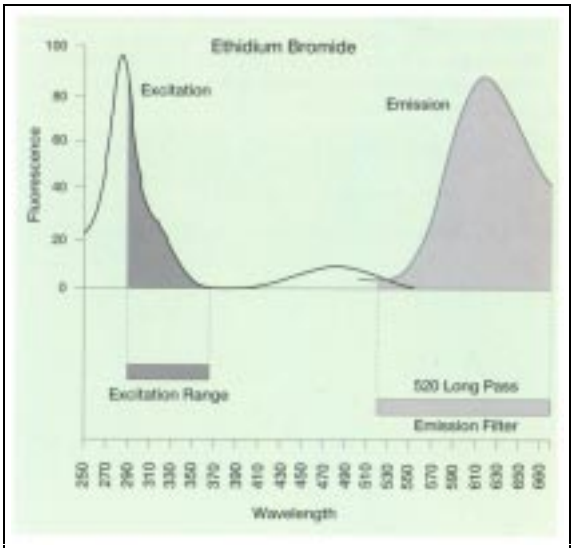


Abbildung 7-6: Anregung und Detektion von Ethidiumbromid.

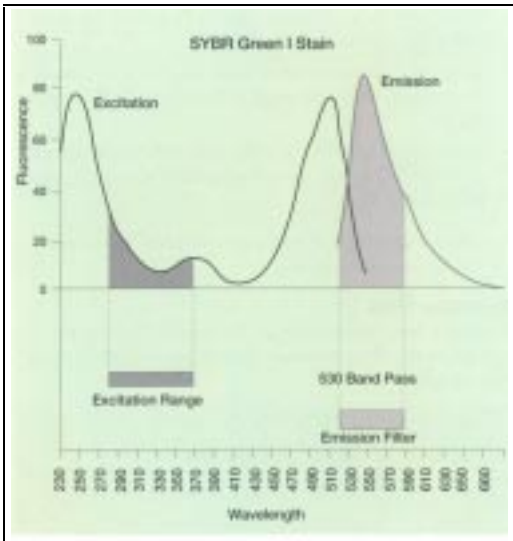


Abbildung 7-7: Anregung und Detektion von SYBR Green I.

DNA-Gehaltes in einer Lösung vereinfacht. Ethidiumbromid färbt hauptsächlich doppelsträngige DNA an. Daneben konnte auch eine geringere Affinität für einzelsträngige DNA und RNA festgestellt werden. In diesem Punkt färbt das SYBR Green I doppelsträngige DNA spezifischer [BIO97]. SYBR Green I kann von Molecular Probes (Eugene, OR) bezogen werden. Somit kann durch Anfärbung und Messung der Fluoreszenz ebenfalls die Konzentration der DNA bestimmt werden.

7.3.3 Elektrophorese

Mit Hilfe der Agarose-Gelelektrophorese läßt sich leicht bestimmen, ob ein Plasmid ein DNA-Insert beinhaltet oder nicht. Das Plasmid mit Insert ist größer und wird daher im elektrischen Feld langsamer durch das Gel bewegt. Über eine digitale Bildanalyse kann nach dem entsprechenden Bandenmuster im Gel gesucht werden. Um nun möglichst effizient alle 96 Plasmid-Präparationen zu untersuchen, müssen diese gleichzeitig auf dem Gel aufgetragen werden. Die Elektrophorese sollte nur so lange ablaufen wie notwendig ist, um die Größenunter-

schiede zwischen Plasmiden mit und ohne Insert eindeutig darzustellen. Der Gesamtablauf der automatischen Präparation sollte durch die gelelektrophoretische Qualitätskontrolle nicht zu sehr verzögert werden, damit der Probendurchsatz auf möglichst hohem Niveau verbleiben kann. Ein Nachteil der gelelektrophoretischen Untersuchung ist, daß sich diese Methode nur unter Bewältigung besonderer Schwierigkeiten in eine automatisierte Umgebung integrieren läßt. So ist der Wiederverwendbarkeit der Gele Grenzen gesetzt. Der Austausch eines „gebrauchten" Gels durch ein neues kann in einer automatisierten Umgebung ein erhebliches Problem werden. Zum einen müssen die Gele vor Gebrauch kühl gelagert werden, zum anderen dürfen sie nicht austrocknen. Der sichere Transport innerhalb der Anlage gestaltet sich aufgrund der Konsistenz des Gels schwierig. Auch die korrekte Handhabung der notwendigen Pufferkammer während der Automation stellen besondere Anforderungen an das zu konstruierende System. Somit scheint die Gelelektrophorese als Routinemethode in einer automatisierten Umgebung weniger gut geeignet zu sein.

7.3.4 Säulenchromatographie

Wesentlich effizienter als die Gelelektrophorese ließe sich ein säulenchromatographisches Verfahren in eine automatisch arbeitende Anlage integrieren. TRANSGENOMIC [Tra97] bietet mit den DNASep™ Säulen Medien an, die in ihren Trennungseigenschaften denen zu vergleichbaren Zwecken eingesetzten gelelektrophoretischen Methoden entsprechen [Hub95]. Die Trennung hat die gleiche Auflösung wie die der Agarose-Gelelektrophorese, ist jedoch um den Faktor 20 schneller durchzuführen. Desweiteren können die Säulen für mehrere tausend Injektionen verwendet werden, bevor das Säulenmaterial auszutauschen ist [Tra97]. Die Säulen können hochparallel betrieben werden, sodaß der Gesamtprozeß durch die zwischen Plasmid-Präparation und PCR-Sequenzierung stattfindende Qualitätskontrolle nicht wesentlich verlangsamt wird.
Es können 96 derartige Säulen zu einer Analyseeinheit zusammengesetzt werden. Jeder Säule wird ein eigener Detektor nachgeschaltet, der die entsprechenden Daten an den Prozeßrechner weiterleitet. Dort wird die Eignung der jeweiligen Probe für den Einsatz in der noch bevorstehenden PCR beurteilt. Eine Dokumentation der Präparationsqualität ist auf diese Weise für den gesamten Probenumfang leicht möglich.

8 Literaturverzeichnis

[Ale77]Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S. A pattern language. Oxford University Press, N. Y., 1977.

[Bai88] Bains, W. und Smith, G. (1988). A novel method for nucleic acid se quence determination. J. Theor. Biol. 135, 303-307.

[Bai93] Bains, W. (1993). Mixed hybridization and conventional strategies for DNA sequencing. Genet. Anal. Tech. Appl. 10, 84-94.

[Ban83] Bankier, A., T., Barell, B., G., Shotgun DNA sequencing. In: Flavell, R., A., et al. (Hrsg.). Techniques in life sciences, B5: Nucleic acid bio chemistry, pp. 1-34. Elsevier-North-Holland, Limerick, Ireland, 1983.

[Bio97] Bio-Rad Bulletin 2132 REV A, 1997, #96-5202 0497, Fluor- S^{TM} Mul tiImager Application Guide. Biorad, Heidemannstr. 164, 80939 München

[Bir79] Birnboim, H., C., Doly, J. (1979), A rapid alkaline extraction procedure for screening recombinant plasmid DNA,Nucl.Acids Res. 7, 1513-1522.

[Blo92] Bloomstein, T., M., Ehrlich, D., J., Laser chemical three dimensional writing form microelectromechanics and applicaton to standard-cell microfluidics. J. Vac. Sci. Technol. 10: 2671, 1992.

[Boo94] Grady Booch, James Rumbaugh: „Object Oriented Analysis and Design. With Applications"; Benjamin/Cummings Publishing Company, Inc, 1994.

[Boo95] Grady Booch, James Rumbaugh: „Unified Method for Object-Oriented Development. Documentation Set"; Rational Software Corporation, 1995.

[Bro75] Brooks, F. 1975. The Mythical Man-Month. Reading, MA: Addison-Wesley.

[Bro87] Brooks, F. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer vol.20(4)

[Bur95] Burns, M.A., Sammarco, T.S., Johnson, B.N., Webster, J., Jones, D., Foerster, B., Fields, Y., Kaiser, A.R., Burke, D., T., Mastrangelo, C.H. Vortrag auf dem Microfabrication Technology Meeting , September 28-29, 1995, San Francisco, California.

[Cha91] Chan, K., C., Koutny, L., B., Yeung, E., S. On line detection of DNA in gel electrophoresis by ultraviolet absorption utilizing a charge coupled de vice imaging system. Anal. Chem. 63: 746. 1991.

[Che90] Cheng, Y., F., Piccard, R., D., Vo- Dinh, T. Appl Spectrosc. 44:755, 1990

[Chr96] Chrisey, L., A., O'Ferral, C., E., Spargo, B., J., Dulcey, C., S. und Cal ver,J., M. (1996). Fabrication of patterned DNA surfaces. Nucleic Acids Res. 1996, 24:15, 3040-3047.

[Cla93]Clark, J., D. OLE und DDE für Windows Programmierer. IWT-Verlag GmbH, Vaterstetten bei München.

[Coa90]Coad, P., Yourdon, E. Object oriented analysis, Prentice Hall, Engle wood Cliffs, N.J., 1990.

[Cox91] Cox, B., J., Novobilski, A., J. Object oriented programming, an evol tionary approach. Second Edition, Addison Wesley, 1991.

[Cra90] Craig, A., Nizetic, D., Hoheisel, J.D., Zehetner, G. und Lehrach, H. (1990). Ordering of cosmid clones covering the Herpes Simplex Vi rus Type I (HSV-I) genome: a test case for fingerprinting by hybridization. Nucleic Acids Res. 18, 2653-2660.

[CRS94] RAPL-II Programming Manual. R-UMI-17-555. CRS Plus Inc. Har rington Court, Burlington Ontario, L7N 3N4 Canada.

[Cul92] Culwin, F.: ADA, a developmental approach. Prentice Hall 1992-1996.

[Drm 89] Drmanac, R., Labat, I., Brukner, I. und Crkvenjakov, R. (1989). Se quencing of megabase plus DNA by hybridization: theory of the method. Genomics 4, 114-128.

[Drm93] Drmanac, R.D., Crkvenjakov, R.B.. (1993). Method of sequencing of genomes by hybridization of oligonucletide probes. US Patent Application 07/838, 607.

[Dut94] Dutton, G. Firms move DNA probes toward more applications in clinic. Genetic Engineering News 14: 1, 1994.

[Egg94] Eggers, M., Hogan M., Reich, R., K., Lamture, J., Ehrlich, D., Hollis, M., Kosicki, B., Powdrill, T., Beattie, K., Smith, S., Varma, R., Ganzadharan, R., Mallik, A., Burke, B., Wallace, D. A microchip for quantitative detection of molecules utilizing luminescent and radioisotope reporter groups. Biotechniques 17(3):516-525, 1994.

[Eng88] Engelke, D.,R., Hoener, P.,A. und Collins, F.,S. (1988). Proc. Natl. Acad. Sci. USA, 85, 544-548.

[Fer97] Konstruktionsbüro Ferchau, Niederlassung Brauschweig.

[Fod91] Fodor, S., P., A., Read, J.L., Pirrung, M.C., Stryer, L., Lu, A.T, und Solas, D. (1991). Light-directed spatially addressabe parallel chemical synthesis. Science 251, 767-773.

[Fod93] Fodor, S., P., A., Rava, R., P., Huang, X., C., Pease, A., C., Holmes,C., P., Adams, C.,P. Multiplexed biochemical assays with biological chips. Nature 364:555, 1993.

[Gam95] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: „Design Patterns“; Professional Computing; Addison Wesley 1995.

[Gam96] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: „Entwurfsmuster“; Professional Computing; Addison Wesley 1995.

[Gen91] Van Genuchten, M. Why is software late ? An empirical study of reasons for delay in software development. IEEE Transactions on software engineering, Vol. 17 (6), June 1991.

[Gol83] Goldberg, A., Robson, D: Smalltalk-80: The Language and its Imple mentation. Reading, Mass: Addison-Wesley, 1983.

[Gre92] McGregor, D., A., Yeung, E., S., Interactive control of pulsed field gel electrophoresis via real time monitoring. Anal. Chem. 64: 1, 1992.

[Gri77] Grimsehl, E., Schallreuter, W., Altenburg, K.: Lehrbuch der Physik, BSB, B.G. Teubner Verlagsgesellschaft, Leipzig. 1977.

[Gui87] Guindon, R., Krasner, H und Curtis, B. 1987, Breakdowns and Process During the Early Activities of Software Design by Professionals. Empirical Studies of Programmers, Second Workshop. Norwood, NewJersey: Ablex Publishing Company.

[Gyl88] Gyllensten, U., B., und Ehrlich, H., A., (1988), Proc. Natl. Acad. Sci. USA, 87, 658-662.

[Har92] Harrison, D., J., Manz, A., Fan, Z., Ludi, H., Widmer, H., M., Capil lary electrophoresis and sample injection systems integrated on a planar glass chip. Anal. Chem. 64:1926, 1992.

[Hic93] Hickersberger, A. Der Weg zur Objektorientierten Software. Hüthig Verlag.1993

[Har93] Harrison, D., J., Fluri, K., Seiler, K., Fan, Z., Effenhauser, C., S., Manz, A., Micromachining a miniaturized capillary electrophoresis based chemical analysis system on a chip. Science 261: 895, 1993.

[Hoh93] Hoheisel, J. D., Maier, E., Mott, R., McCarthy, L. Grigoriev, A. V., Schalkwyk, L.C., Nizetic, D., Francis, F. und Lehrach, H. (1993). High re solution cosmid and P1 maps spanning the 14 Mbp genome of the fission yeast *Schizosaccharomyces pombe*. Cell 73, 109-120.

[Hoh96] Hoheisel, J. D. (1996). Sequence-independent and linear variation of oligonucleotide DNA-binding stabilities. Nucleic Acids Res. 24, 430-432.

[Hub95] Huber, C., G., Oefner, P., J., Bonn, G., K., Accurate sizing of DNA fragments by ion-pair reversed-phase chromatography. Anal. Chem (1995), 67, 578-585.

[Inn88] Innis, M., A., Myambo, K., B., Gelfand, D., H., und Brow, M., A., D., (1988), Proc. Natl. Acad. Sci. USA, 85, 9436-9440.

[Iso92] Qualitätsmanagement- und Qualitätssicherungsnormen. Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung, Lieferung und Wartung von Software (Identisch mit ISO 9000-3:1991), Beuth Verlag, Juni 1992.

[Iso94] Qualitätsmanagementsysteme. Modell zur Qualitätssicherung/QM-Darlegung in Design, Entwicklung, Produktion, Montage und Wartung, Beuth Verlag, August 1994.

[Jac88] Jackson P., Urwin, V., E., Mackey, C., D. Rapid imaging using a cooled charge coupled device of fluorescent twodimensional polyacrylamide gels

produced by labeling proteins in the first dimensional isoelectric focusing gels with the fluorophore 2-methoxy-2, 4-diphenyl-3 (2H) furane Electro phoresis 9:330, 1988.

[Jon92] Jones, C. 1992. Risky Business: The most common software risks. Ameican Programmer, Vol. 5(7)

[Kar91] Karger, A., E., Harris, J., M., Gesteland, R., F. Multiwavelength fluorescence detection for DNA sequencing using capillary electrophoresis. Nuc. Acids Res. 19: 4955, 1991.

[Khr89] Khrapko, K., R., Lysov, Y., P., Khorlin, Shick, V., Florentiev, V., L. und Mirzabekov, A., D. (1989). AN oligonucleotide hybridization approach to DNA sequencing. FEBS Lett 256, 118-122.

[Khr91] Khrapko, K., R., Lysov, Y., P., Khorlin, A., A., Ivanov, I., B., Yershov, G., M., Vasilenko, S., K., Florentiev, V., L. und Mirzabekov, A., D. (1991). A method for DNA sequenzing by hybridization with oligo nucleotide matrix. DNA Sequence 1, 375-388.

[Kee89] Keene, S: Object Oriented Programming in Common Lisp: A Pro grammers Guide to CLOS. Reading, Mass.: Addison-Wesley, 1989.

[Klu88] Klunder, D.: Naming Conventions (Hungarian). White Papers, Microsoft Press. January 18, 1988.

[Koo93] Koop, B., F., Rowan, L., Chen, W., -Q., Deshpande, P., Lee, H., Hood, L. (1993), BioTechniques 14, 442-447.

[Kos92] Kostichka, A., J., Marchbanks, M., L., Brumley, R., L., Jr., Drossman, H., Smith, L., M. High speed automated DNA sequencing in ultrathin slab gels. Biotechnology 10: 78, 1992.

[Leh94] Lehrach, H., Maier, E., Meier-Ewert, S., Ahmadi, A., R., Curtis, J. Application of robotic technology to automated sequence fingerprint analy sis by oligonucleotide hybridisation. Journal of biotechnology 35 (1994), 191-203.

[Mac89] Mackay, E.: Analysis of Samples by Electrophoresis using a charge coupled device, US Patent: 4,874,492, October 17, 1989.

[Mag93] Maguire, S., A., Writing Solid Code. Microsoft Press, Redmond, Washington 1993.

[Man92] Manz, A, Harrison, D., J., Verpoorte E., M., J., Fettinger, J., C., Pau lus, A., Ludi H, Widmer H., M. Capillary electrophoresis on a chip, J. Chromatography 593:253,

[Mas92] Maskos, U., Southern, E. (1992). Parallel analysis of oligodeoxyribo nucleotide interactions. I. Analysis of factors influencing oligonucleotide duplex formation. Nucleic Acids Res. 20, 1675-1679.

[Mat95] Matson, R., S., Rampal, J., Pentoney, S., L., Anderson, P., D., Coassin, P. (1995). Biopolymer synthesis on polypropylene support: Oligonucleotide arrays. Anal. Biochem. 224, 110-116.

[Mey85] Meyerowitz, E., M., Pruitt, R., E., *Arabidopsis thaliana* and plant mo lecular genetics. Science 229:1214-1218 (1985)

[Mey88] Meyer, B.:Genericity versus inheritance. OOPSLA'86 as ACM SIGPLAN 21, 11 (Nov. 1986), 391-405.

[Mic88] Micallef, J: Encapsulation, Reusability and extensibility in object-oriented languages. Journal of Object Oriented Programming 1, 1 (April/May 1988), 12-38.

[Mir94] Mirzabekov, A., D. (1994). DNA sequencing by hybridization- a me gasequencing method and a diagnostic tool. Trends Biotechnol. 12, 27-32.

[Mul87] Mullis, K., B., und Faloona, F., A. (1987). In R. Wu, (ed) Methods Enzymol. 155, 335-350, Academic Press, London.

[Nan95] Nanogen, Inc. 10398 Pacific Center Court, San Diego, Ca 92121: Heller, M., J., Vortrag auf dem Microfabrication Technology Meeting , September 28-29, 1995, San Francisco, California.

[Nor93] Northrup, M., A., Ching, M., T., White, R., M., Watson, R., T., Proceedings of the Seventh International Conference on Solid State Sensors and Actuators, Yokohama, Japan, June 6-10, 1993, pp. 924-926.

[Pac90] Pace, S., J., Silicon Semiconductor Wafer for Analyzing Micronic Biological Samples, U.S. Patent Number 4,908,112, March 13, 1990.

[Pan87] Pang, P., P., Meyerowitz, E., M. *Arabidopsis thaliana*: a model system for plant molecular biology. Bio/Technology 5; 1177-1181 (1987)

[Pea94] Pease, A., C., Solas, D., Sullivan, E., J., Cronin, M., T., Holmes, C., P., und Fodor, S., P., A. (1994). Light generated oligonucleotide arrays for rapid DNA sequence analysis. Proc. Natl. Acad. Sci. USA, 91, 5022-5026.

[Pen92] Pentoney, Jr. et al. Detection of Radioisotope Labeled Components in a Capillary. US Patent 5,143,850, September 1 ,1992

[Pet82] Petersen K., E., Silicon as a mechanical material. Proc. IEEE 70:420-457, 1982.

[Qia95] Qiaprep Plasmid Handbook, March 1995, Qiagen GmbH and Qiagen Inc., 1995.

[Qia97] Qiaprep Miniprep Handbook, April 1997, Qiagen GmbH and Qiagen Inc., 1997.

[Row97] Rowen, L., Mahairas, G., Hood, L.: „Sequencing the Human Genome" (1997), Science 278, 605-607.

[Rum91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: „Object Oriented Modeling and Design"; Prentice Hall 1991.

[Sai88] Saiki, R., K., Gelfand, D.H., Stoffel, S. Scharf, S.J., Higuchi, R., Horn, G., T., Mullis, K., B. und Erlich, H., A. (1988) Science 239, 487-491.

[Sch86] Schaffert, C., Cooper, T., Bullis B., Kilian, M., Wilpolt C.: An intro ducton to Trellis-Owl. . OOPSLA'89 as ACM SIGPLAN 21, 11 (Nov. 1986), 9-16.

[Sei93] Seiler, K., Harrison, D.,J., Manz, A. Planar glass chips for capillary electrophoresis: repetitive sample injection, quantitation and separation ef ficiency. Analy Chem 65:1481, 1993.

[Sha89] Shah, A., Rumbaugh, J., Hamel, J., Borsari, R: DSM: an object-relationship modeling language. OOPSLA'89 as ACM SIGPLAN 24, 11 (Nov. 1989), 191-202.

[Shl88] Shlaer, S., Mellor, S., J. Object oriented system analysis, Yourdon, Englewood Cliffs, N., J., 1988.

[Sou92] Southern, E., Maskos, U. und Elder, J., K., (1992). Analysing and comparing nucleic acid sequences by hybridization to arrays of oligo nucleotides: evaluation using experimental models. Genomics 13, 1008-1017.

[Sti94] Stix, G. Gene readers, Scientific American 270: 149, 1994.

[Str92] Stroustrup, B: Die C++ Programmiersprache. Addison Wesley Verlag. 1992

[Swe91] Sweedler, J., V., Shear, J., B., Fishman, H., A., Zare, R., N., Scheller, R., H. Fluorescence detection in capillary zone electrophoresis using a charge coupled device wit time delayed integration. Anal. Chem 63:496, 1991.

[Tav90] Tavrow, L.,S., Bart, S., F., Lang, J., H., Sens Actuators A, 35:33, 1992.

[Tra97] Transgenomic inc. http://www.transgenomic.com

[Ver91] Verpoorte E., Manz, A., Ludi, H., Widmer, H., M., van der Schoot, B., H., de Rooij, N., F., A novel optical detector cell for use in miniaturized total analysis systems, Proceedings of Transducers 1991, San Francisco, Sept.15-19, 1991, pp 796-799.

[Wis91] Wise, K., D., Najafi, K. Microfabrication techniques for integrated sensors and microsystems. Science 254:1335, 1991.

[Woo94] Woolley, A., T., Mathies, R., A., (1994). Ultra high speed DNA-fragment separations using microfabricated capillary array electrophoresis chips. Proc. Natl. Acad. Sci., USA, Vol. 91, 11348-11352, Nov. 1994. Bio physiscs.

[Yam91] Yamamoto, T., Masakazu, N., Masayoshi, M., Electrophoresis pattern analyzer for genetic material, US Patent 5,061,067, October 29, 1991.

[You92] Young, D.,A.: Object Oriented Programming with C++ and OSF-Motif; Prentice Hall 1992

9 Anhang A

9.1 Listing „Simulationsprogramm Virtueller Roboter“

9.1.1 Header Dateien

Controller.h,
ControllerInterface.h,
Declarations.h,
Grammar.h,
Language.h,
Model.h,
Robot.h,
RobotDevice.h,
Swatch.h,
Translator.h,
VacModel.h,
VacuumChamber.h,
VcDevice.h,
View.h,
Vocabulary.h

9.1.2 Source Dateien

ControllerInterface.cpp,
Grammar.cpp,
Language.cpp,
Main.cpp,
Model.cpp,
Robot.cpp,
RobotDevice.cpp,
Translator.cpp,
VacModel.cpp,
VacuumChamber.cpp,
VcDevice.cpp,
View.cpp,
Vocabulary.cpp

9.1.3 Konfigurationsdatei.

VC.CFG

9.1.2 Header Dateien.

```
/*~-------------------------------------------------------------------------------------------------------------*\
| Controller: Managing the abstract machine.                                                                    |
|                                                                                                               |
|  File: controller.h                                                                                           |
|  Running level in design of robot controller software: 0                                                      |
|  Task:                                                                                                        |
|           Installing a temporary relation to the translator to get the tasks for the concrete machine. Instantiation of the |
|           Views and Model. Definition of an interface (see pure virtuals for explanation). All in all this class is not pure|
|           virtual to make sure that this controller is not used as an interface d efinition, only. Moreover, the pure inter- |
|           face for that purpose is defined in the class "ControllerInterface" (see file controllerinterface.h for expla-    |
|           nations.)                                                                                           |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Environment for Development:                                                                                  |
|                                                                                                               |
|                   Intel microprocessor Pentium 133, Windows NT, Version 4.0, Microsoft Developer Studio for C++, Version 4.0
|                                                                                                               |
| Interface:                                                                                                    |
|                                                                                                               |
|                   Microsoft Windows, shipped with NT, Version 3.51, Tested with Windows shipped with NT Version 4.0.
|                                                                                                               |
| Operating System:                                                                                             |
|                   Windows NT,Version 3.51 German Language Environment.                                        |
| Author:                                                                                                       |
|                   Gerhard Kauer, GBF-Braunschweig, Dpt. GNA (Genome Analysis), Mascheroder Weg 138124 Braunschweig, Germany
|                                                                                                               |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Rules for layout used:                                                                                        |
|                                                                                                               |
|                   1. Functions do begin with uppercase letters and may contain other uppercase letters, too.  |
|                                                                                                               |
|                   2. Variables do begin with lowercase letters and may contain uppercase letters for better readabil ity.       |
|                   3. "Class globals" do identify with an beginning underscore: e.g. _iInteger;               |
|                   4. Directories do begin with uppercase letters and may contain other uppercase letters for better readabil i-
ty.|
|                   5. Filenames do not contain uppercase letters.                                              |
|                   6. Classnames do begin with uppercase letters and may contain other uppercase letters, too. |
|
```

Extensions for the hungarian notation format used with this documentation:

1. Pointers:
   - 1.1. Pointers to classes := 'cptr' .
   - 1.2. Pointers to a data structure type := 'stptr'.

2. Variables:
   - 2.1. Void members := 'v' .
   - 2.2. Data structures := 'st' .
   - 2.3. Classes := 'cl' .

3. Adress variables for functions:
   - 3.1. Callbacks := 'cb' .

4. Classes
   - 4.1. Simple instances of a class := 'cl' .

5. Streams
   - 5.1. Files := 'sm' .

6. Templates
   - 6.1. Pointers := 'tptr' .
   - 6.2. Variables := 't' .

```
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-----------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_CONTROLLER
 #define ROBOT_CONTROLLER
 #include "declarations.h"
 class Controller {
       public:
                   Controller(int iArgs,char **szArgs);                    // A command line vector system is used as input.
                   ~Controller(void);                                              // Controlled destruction of allocated
resources.
                   int TranslateArguments(void);                                   // Translate concrete commands to virtual
manchine ones.
                   char ClassReadyToUse(void);                                     // Indicate complete instantiation and
initialization.
                   virtual void Run(void)=0;                                       // Make the machine robot work.
                   LanguagePackage GetLanguagePackage(void);               // Interface funtion for using the virtual language.
                   TaskSchedule GetTaskPackage(void);                              // Interface function to stick to the concre-
te tasks.
                   ActionList *GetCurrentAction(void);                             // Interface function to get the current
task.
       private:
                   char **_szArgs;                                                       // Save the argument vector as
the only input source...
                   int _iArgs;                                                           // ... and fix it’s size.
                   char _bClassReadyToUse;                                         // Flag returned with-
>ClassReadyToUse(); Interface.

       protected:
                long int GetNumerical(char *szLine,int iPos);         // Auxiliary: Excerpt numerical data from ascii streams.
                   virtual char TestInfoLine(void);                        // Interface: Get control on the input vector.
                   virtual char TestGrammar(void)=0;                               // Interface: Get control on the grammar com-
piled.
                   virtual char TestVocabulary(void)=0;                    // Interface: Get control on the vocabulary used.
                   virtual char GenerateApplicationParameters(void)=0; // Interface: Build task schedule: combine tasks + devi-
ces.
                   virtual char TestApplicationParameters(void)=0;         // Interface: Get control on the compiled task sche-
dule.
                   virtual char LaunchModel(void)=0;                               // Interface: Instantiate the model of the
virtual machine.
```

```
                    virtual char LaunchViews(void)=0;                                   // Interface: Instantiate the views of the
virtual machine.
                    virtual void ConfigureModel(void)=0;                        // Interface: Download parms to the model.
                    virtual void InvalidateViews(void)=0;                       // Interface: Cause connected views to recall.
                    virtual void InvalidateViews(long int)=0;                   // Interface: Cause connected views to recall with
parm.
                    int GetAmountArguments(void);                                       // Interface: Allow external access to the
input ...
                    char **GetArgumentVector(void);                                     // ... argument vector stored with this in-
stance.
                    LanguagePackage _stLanguage;                                        // Struct containing compiled Grammar and
Vocabulary.
                    TaskSchedule _stTaskSchedule;                                       // Struct containing a list of device events.
                    ActionList *_stActionList;                                                  // Struct containing a list of device
actions.
 };
#endif
/*~-------------------------------------------------------------------------------------------------------------------*\
| ControllerInterface: Entail an interface for control purposes.                                                      |
|                                                                                                                     |
|  File: controllerinterface.h                                                                                        |
|  Running level in design of robot controller software: 0                                                            |
|  Task:                                                                                                              |
|            The model class and the view class should be accessible with the same i nterface. With that strategy, there is an  |
|            easy way to exchange descendants within the controller. Advantage with Patterns like "Abstract Factories" and     |
|            "Strategy Patterns". The interface is pure virtual, therefore. It’s task is to entail an interface.
        |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                             |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_CONTROLLER_INTERFACE
 #define ROBOT_CONTROLLER_INTERFACE
 class ControllerInterface {
       public:
                    ControllerInterface(void);            // Initialization of the controller class is secondary with a class ...
                    ~ControllerInterface(void);           // ... which task is to entail an interface, only.
```

```
                    virtual void Invalidate(void)=0; // Interface: Inform descendant, that it’s current understanding is out of
date.
                    virtual void Invalidate(long)=0;       // Interface: Overloaded version which allows to pass a unit.
                    virtual char ClassReadyToUse(void)=0;// Interface: access to readyness (important for further dev elopment).
      protected:
                    virtual void Synchronize(void)=0;// Interface: Do a recall. Get the current information from Contro ller.
 };

#endif
#define VERBOSE 1         // Needed during development to get some information.
#define SIMULATION 1      // Doing simulations, only. Gateways will not be used.

#define ROB_LPT1  1
#define ROB_LPT2  2
#define ROB_COM1  3
#define ROB_COM2  4
#define ROB_DDE   5
#define DEV_PORT  1
#define DEV_DDE   2
#define DEV_NET   3
#define TRUE 1
#define FALSE 0
#define MAX_CFGFILESIZE 10000
//------------------------------------ Profiling file system ------------------
#define PROFILING_FILE_SYSTEM_GRAMMAR_INDICATOR "GRAMMAR:"
#define PROFILING_FILE_SYSTEM_ACTION_INDICATOR "ACTIONS:"
#define GRAMMAR_SEPARATOR_INDICATOR "SEP"
#define GRAMMAR_TIME_INDICATOR "TIM"
#define GRAMMAR_CYCLING_INDICATOR "CYC"
#define GRAMMAR_PARENTHESIS_INDICATOR "PAR"
#define GRAMMAR_ENUMERATOR_INDICATOR "ENU"
#define GRAMMAR_DEVICE_INDICATOR "DEV"
#define GRAMMAR_COMMENT_INDICATOR "COM"
#define GRAMMAR_IDENTPOSITION 4
#define ASCII_TAB 9
#define MAXLONGINT 0xFFFFFFFF
#define DEVICE_EVENT_TYPE_TIMEDEPENDEND 1
#define DEVICE_EVENT_TYPE_TEMPERATUREDEPENDEND 2
#ifndef GRAMMAR_STRUCT
#define GRAMMAR_STRUCT
 typedef struct _GRAMMAR {
  char _bSeparator;
  char _bCycleSymbol;
  char _bOpenParenthesis;
  char _bCloseParenthesis;
  char _bEnumerator;
  char _szCommentSymbol[3];
```

```
  char *_bDeviceSymbol;
  char _bTimeSymbol;
  int _iDevices;
 }GrammarPackage;
#endif
#ifndef VOCABULARY_STRUCT
#define VOCABULARY_STRUCT
 typedef struct _VOCABULARY {
  int _iNumVocListElements;
  char **_szVocList;
 }VocabularyPackage;
#endif
#ifndef LANGUAGE_STRUCT
#define LANGUAGE_STRUCT
 typedef struct _LANGUAGE {
  char bStructValid;
  GrammarPackage stGrammar;
  VocabularyPackage stVocabulary;
 }LanguagePackage;
#endif
#ifndef DEVICE_EVENT_STRUCT
 #define DEVICE_EVENT_STRUCT
 typedef struct _DEVICE_EVENT {
       int iDevice;
       char bEventType;                        // Time oriented, Temperature, etc.
       long int lStartActionTick;              // With periodically actions: 0, else absolute tick-value.
       long int lStopActionTick;            // With periodically actions: 1, else absolute tick-value.
       long int lCyclingPeriod;             // e.g. 5 for "open for 1 tick after all 5 ticks.
 }DeviceEvent;
#endif
#ifndef TASK_SCHEDULE_STRUCT
 #define TASK_SCHEDULE_STRUCT
 typedef struct _TASK_SCHEDULE {
       int iEvents;
       DeviceEvent *stDeviceEvent;
 }TaskSchedule;
#endif
#ifndef DEVICE_ACTION
 #define DEVICE_ACTION
 typedef struct _DEVICE_ACTION {
  int iDevice;
  char bActive;
 }DeviceAction;
#endif
#ifndef TASK_LIST_STRUCT
 #define TASK_LIST_STRUCT
 typedef struct _ACTION_LIST {
```

```
      int iActions;
      DeviceAction *stDeviceAction;
 }ActionList;
#endif
#ifndef DEVICES_PARAMETER
 #define DEVICES_PARAMETER
 typedef struct _DEVICES_PARAMS {
      int iPort;
      char bTypeOfDevice;       // PORT, DDE (Server)
 }DevParams;
#endif
/*~-------------------------------------------------------------------------------------------------------------------*\
| Grammar: Compiling a grammar package.
                                                |
|
|
|  File: grammar.h
|
|  Running level in design of robot controller software: 0                                                            |
|  Task:                                                                                                              |
|            Compiling a grammar list which contains the rules with which commands within a configuration file will set up the |
|            corresponding action list of an abstract machine. Finally this grammar will be used together with a given vocabu- |
|            lary to compose a language which can be used to translate the given tasks into the abstract tasklist of the ab-   |
|            stract machine.    A given configuration file, which contains the grammar to be used, will be investigated. The exe-
|
|            cutable grammar elements will be compiled to a final package of encaps ulated grammar rules (GrammarPackage).    |
|            This result will be accessible with an interface function, then.
                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                             |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_GRAMMAR
 #define ROBOT_GRAMMAR
 #include "declarations.h"
class Grammar {
      public:
                  Grammar(void);                                    // Initializations and standardization of member vars.
                  ~Grammar(void);                                   // Controlled destruction of allocated resources.
```

```
                  void ReadGrammarList(char *szFile); // Get the grammar paragraph from a given configur ation file.
                  void GenerateGrammar(void);                 // Download the given rules to a structure managing the grammar.
                  GrammarPackage GetGrammar(void);// Interface: Make the structure accessible as grammar package.
                  char ClassReadyToUse(void);                                          // Indicate complete instantiation and
initialization.
      private:
                  GrammarPackage _stGrammar;                  // Encapsulate the rules compiled.
                  char _bClassReadyToUse;                     // Flag returned with->ClassReadyToUse(); Interface.
      protected:
                  char **_szCmdList;                          // 2 dimensional list of executable grammar elements. These el e-
ments ...
                  int _iNumCmdListElements;             // ... will be identified and compiled to a final package of grammar
rules.
                  void AddToCmdList(char *szCmd);       // Auxiliary functionality: Enlarging a 2-dim list by one element.
 };

#endif
/*~------------------------------------------------------------------------------------------------------------------*\
| Language: Mixin class, unifying the two members of a language:  grammar and vocabulary.
        |
|
|
|  File: language.h
                                                  |
|  Running level in design of robot controller software: 2                                                           |
|  Task:                                                                                                             |
|           Unifying grammar and vocabulary of an abstract language into one accessible language package.
        |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Environment for Development:
|
|                  Intel microprocessor Pentium 133, Windows NT, Version 4.0, Microsoft Developer Studio for C++, Version 4.0
|
| Interface:
|
|                  Microsoft Windows, shipped with NT, Version 3.51, Tested with Wi ndows shipped with NT Version 4.0.
|
| Operating System:
                                                  |
|                  Windows NT,Version 3.51 German Language Environment.
                                |
| Author:
                                                    |
|                  Gerhard Kauer, GBF-Braunschweig, Dpt. GNA (Genome Analysis), Mascheroder Weg 1 38124 Braunschweig, Germany
|
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                           |
```

```
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*------------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_LANGUAGE
 #define ROBOT_LANGUAGE


 #include "grammar.h"
 #include "vocabulary.h"
 class Language : public Grammar, public Vocabulary {
      public:
                  Language(void);                                           // Instantiation of the mixin class may need some
initialization.
                  ~Language(void);                                    // Controlled destruction of allocated resources.
                  LanguagePackage GetLanguage(void);           // Interface: Unifying vocabulary and grammar into one language
...
                                                                                    // ... package. Make this encapsulated packa-
ge accessible to aliens.
      private:
                  LanguagePackage _stLanguage;                 // Encapsulated structure containing vocabulary and grammar of a
language.
 };

#endif
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Model: Implementation of a standard model, which can be inherited to the concrete machines.
        |
|
|
|  File: model.h
                                                      |
|  Running level in design of robot controller software: 1                                                      |
|  Task:                                                                                                        |
|            The model class is a base class for concrete machine (dependend) models. It excerps all the parts which may be    |
|            found in all models and therefore stand per definitionem for the expression of "model".    The anchor class is sol-
ved|
|            here and the abstract task list is managed, too.
                                  |
| History:
|
```

```
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-----------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_MODEL
 #define ROBOT_MODEL
 #include "controllerinterface.h"
 #include "controller.h"
 class Model : public ControllerInterface {
       public:
                   Model(void *cptrAnchor);                                    // Receive anchor class and do some initializations.
                 virtual void * GetCurrentTaskList(long int)=0;  // Interface: Allow reading the task list.
                   ~Model(void);                                                          // Controlled destruction of allocated
resources.
     protected:
                   TaskSchedule _stTaskList;                                   // Struct containing a list of device events.
                   Controller *_cptrAnchor;                                    // Anchor class, which offers an informational in-
terface.
                   long int *_slCycleInterruptor;                              // Counter for interruption of cyclic actions like -
V3tc(5,1)
 };
#endif
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Robot: Implement the controller event dependend actions for an abstract robot device.
               |
|
|
|  File: robot.h                                                                                                      |
|  Running level in design of robot controller software: 2                                                            |
|  Task:                                                                                                              |
|            See view.h; To solve hardware-dependency, the abstract robot will get a ccess to a concrete machine (RobotDevice)  |
|            which is an object of it's own.
                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_ROBOT
 #define ROBOT_ROBOT
 #include "view.h"
 #include "robotdevice.h"
 class Robot : public View {
       public:
                   Robot(void *,RobotDevice *);                 // Passed: anchor class and robot device. Doing initializ ations.
                   ~Robot(void);                                          // Controlled destruction of allocated resou rces.
                   void Invalidate(void);                              // Interface: Solving pure virt ual::ControllerInterface.
                   void Invalidate(long);                              // Interface: Solving pure virt ual::ControllerInterface.
                   char ClassReadyToUse(void);                         // Interface: Indicate complete instantiation and initi a-
lization.
       private:
                   char _bClassReadyToUse;                             // Flag returned with ->ClassReadyToUse():Interface;
                   long int _lCurrentUnit;                             // Keep current controller-unit (sent with the last inv a-
lidation signal).
                   RobotDevice *_cptrMachine;                          // Concrete machine instaniated separately (most often in
Controller).
                   void Synchronize(void);                             // Solving pure virtual::ControllerInterface.
                   void Operation(ActionList *);                // Transfer generated action list to the concrete devices.
};


#endif
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| RobotDevice: Implement the controller event dependend actions for a concrete robot device.
|
|
|
|  File: robotdevice.h
|
|  Running level in design of robot controller software: 0                                                          |
|  Task:                                                                                                            |
|            The concrete device is implemented with a strategy method, so that different concrete robots may be excha nged     |
|            during application's runtime. Therefore all devices should offer a unique interface for their access. This inter- |
|            face is inherited from this base class. To avoid artificial enlargement of inheritance in the overall d esign, that|
|            class is not meant to be abstract. If there is need to fix some uniquely usable routines here, one may do so. The |
|            main task of this class is to entail an interface to all concrete mach ines. All further development touching the  |
|            interface-behaviour of the concrete machine shoul d use this class for that purpose.
|              |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_ROBOT_DEVICE
 #define ROBOT_ROBOT_DEVICE
 #include "declarations.h"
class RobotDevice {
      public:
            RobotDevice(DevParams stParams);                                  // Hardware dependend parameters will be  managed
here.
            ~RobotDevice(void);                                                        // Controlled destruction of allocated
resources.
            virtual void SpecificOperation(ActionList *stList)=0;     // Interface: Adaptor-Class behaviour is needed.
        char RobotDeviceRunning(void);                                        // Interface: Aliens may ask, whether the virtual
robot still works.                                                                                  // Hardware
dependend operation is to be processed here.
      protected:
            char _bOmegaSystemRunning;                                         // Flag indicating whether to stop the
omega system or not.
};

#endif
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Translator: Generate language for abstract machine.
|                                  |
|
|
|  File: translator.h
|                                                      |
|  Running level in design of robot controller software: 0                                                          |
|  Task:                                                                                                            |
|            Generating a list of grammar and vocabulary, which is ready for use with the abstract machine.
|
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_TRANSLATOR
 #define ROBOT_TRANSLATOR
#include "language.h"

 class Translator {
       public:
                   Translator(char *szActionFile);        // Translation is done using a given configuration file.
                   ~Translator(void);                     // Controlled destruction of allocated resou rces.
                   char ReadyToUse(void);                 // Interface: Test the object's init. status before tran slating.
                   char Translate(void);                  // Interface: Cause object to run translation.
                   LanguagePackage GetLanguage(void);     // Interface: Alien objects get access to the encapsulated pack age.
       private:
                   Language *_cptrLanguage;               // The translator needs to have an association with a language object.
                   LanguagePackage _stLanguage;           // Encapsulated language package, containing the abstract langu age.
                   char _bClassReadyToUse;                // Flag returned with ->ReadyToUse():Interface.
                   char *_szActionFile;                   // Configuration file containing machine's acion list.
       protected:
                   void GetActionList(void);              // Generate grammar and vocabulary.
 };
#endif
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| VacModel: Implement the vacuum chamber model.                                                                       |
|                                                                                                                     |
|  File: vacmodel.h                                                                                                   |
|  Running level in design of robot controller software: 2                                                            |
|  Task:                                                                                                              |
|            See model.h; Solving the pure virtual functions of the model class. The machine dependend actions are so lved here |
|            with the concrete model, there will be built up a specific action list, representing the concrete model (world) of|
|            the system supported. Theses actions will cause the concrete machine (RobotDevice) to send the hardware dependend |
|            commands, then.                                                                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:                                                                                                            |
|                                                                                                                     |
|                  Date         Reason                                                                              |
|                                                                                                                     |
|                  15.03.1996   Foundation                                                                          |
|                                                                                                                     |
|                  1.0                                                                                              |
|                                                                                                                     |
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_VAC_MODEL
 #define ROBOT_VAC_MODEL
 #include "model.h"
class VacModel : public Model {
public:

            VacModel(void *cptrAnchor);                         // Initializations and standardization of member vars.
            ~VacModel(void);                                    // Controlled destruction of allocated resources.
            void Invalidate(void);                              // Interface: Solving the pure virtual::ControllerInterface.
            void Invalidate(long);                              // Interface: Solving the pure virtual::ControllerInterface.
            char ClassReadyToUse(void);                         // Interface: Indicate complete instantiation and initialization.
            void * GetCurrentTaskList(long int);   // Interface: Access to the current action list dependend with this model.
protected:
            void Synchronize(void);                                   // Solving the pure virtual::ControllerInterface.
            char AddDeviceAction(int iA);                       // Different devices may be active at the same time, depending on
the current task ...
                                                                                  // ... list. E.g. Valve1 and Pump and Valve3
may be active during "cyclic vacuum destruction".
            ActionList _stActionList;                           // Struct containing a list of device actions (dependend with
this model).
```

```
            char _bClassReadyToUse;                                  // Flag indicating the status of initiation during co n-
struction.
};

#endif
/*~-------------------------------------------------------------------------------------------------------------------*\
| VacuumChamber: Managing the vacuum chamber control.                                                                 |
|                                                                                                                     |
|  File: vacuumchamber.h                                                                                              |
|  Running level in design of robot controller software: 1                                                            |
|  Task:                                                                                                              |
|            See controller.h; Solving the pure virtual functions of the controller class. The machine "Vacuum Chamber" needs  |
|            to implement hardware dependend extentions like valves, pump and port. These logical units are instantiated here. |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_VACUUMCHAMBER_CONTROLLER
 #define ROBOT_VACUUMCHAMBER_CONTROLLER
 #include "controller.h"
 #include "declarations.h"
 #include "vcdevice.h"
 const unsigned char VALVE_1_PSEUDOCODE=1;                           // Declaration of device identifiers ...
 const unsigned char VALVE_2_PSEUDOCODE=4;                           // ... which were set to the binary  ...
 const unsigned char VALVE_3_PSEUDOCODE=8;                           // ... control values of the elec-   ...
 const unsigned char VENTILATOR_PSEUDOCODE=16;                       // ... tronic Vacuum Chamber Con-    ...
 const unsigned char PUMP_PSEUDOCODE=32;                             // ... troller interface responsible ...
 const unsigned char ARRETATION_1_PSEUDOCODE=64;                // ... for this external hardware.   ...
 const unsigned char ARRETATION_2_PSEUDOCODE=128;               // ...
 const unsigned char TERMINATE_OMEGA=254;                            // ... .
 class VacModel;                                                                  // Auxiliary objects, necessary to   ...
 class Swatch;                                                                          // ... build up an Controller Model
...
 class Robot;                                                                           // ... View (CMV) callback system.
 class VacuumChamber : public Controller {
       public:
                   VacuumChamber(int iArgs,char**szArgs);            // A command line vector system is used as i nput.
```

```
                    void Run(void);                                                 // Make the vacuum chamber active.
(Solved pure virtual::Controller)
                    ~VacuumChamber(void);                                       // Controlled destruction of allocated re-
sources.
                    char ClassReadyToUse(void);                                 // Indicate complete instantiation and in-
itialization.
      private:
                    unsigned char _ubValve1;                        // These are the devices which were run with the concrete  ...
                    unsigned char _ubValve2;                        // ... machine. Namely they are the valves for vacuum flow ...
            unsigned char _ubValve3;                        // ... control, the vacuum pump itself, the drying ventilator ...
                    unsigned char _ubPump;                                  // ... system (e.g. to get rid of the remaining alcohol
during...
                    unsigned char _ubVentilator;                    // ... the last step of a Qiagen plasmid preparation) and the ...
                    unsigned char _ubArretationA;                   // ... two locking stations at the Biomek 2000 pipetting robot
...
                    unsigned char _ubArretationB;                   // ... whích were driven by pneumatic force ...
                    unsigned char _ubOmegaRobot;                    // ... The omega robot is the complete conglomerat of devices
used here.
                    char _bClassReadyToUse;                                 // Flag returned with ->ClassReadyToUse(): Interface.

                    char GetDevice(int iElement);                   // Specify the device ordered with the given language.
                    char GetSchedule(int iElement);                 // Distinguish: linear or cycling events using the given langua-
ge.
                    char GetCyclingSchedule(int,int);               // Install a cycling event to the task schedule.
                    char GetTimingSchedule(int,int);                // Install a linear event to the task schedule.

                    VacModel *_cptrVacModel;                        // The vacuum model object needed to implement the action list.
                    Swatch *_cptrSwatch;                                    // The vacuum swatch object needed for simulation and/or
display                                                                                 // purposes.
             Robot *_cptrRobot;                                             // A robot consists of n Devices (Valves, Pump, etc.)
             VcDevice *_cptrVcDevice;                               // Hardware dependend Device "VacuumChamber" as visitor used with
                                                                                        // "Robot".


      protected:
                    void AssignDevices(void);                       // Assigning pseudocodes to concrete devices.
                    char TestGrammar(void);                                 // Get control on compiled grammar.(Solved pure virtu-
al::Controller)
                    char TestVocabulary(void);                              // Get control on compiled vocabulary.(Solved pure virtu-
al::Controller)
                    char GenerateApplicationParameters(void);// Build task schedule. (Solved pure virtual::Controller)
                    char TestApplicationParameters(void);  // Get control on the task schedule. (Solved pure virtual::Controller)
                    void ConfigureModel(void);                              // Download parms to the model. (Solved pure virtu-
al::Controller)
                    char LaunchModel(void);                 // Instantiate vacuum chamber model. (Solved pure virtual::Controller)
                    char LaunchViews(void);                 // Instantiate vacuum chamber views. (Solved pure virtual::Controller)
// Cause views to recall. (Solved pure virtual::Controller)
                    void InvalidateViews(long int);         // Cause views to recall with parm. (Solved pure virtual::Controller)
```

```
 };


#endif

/*~-------------------------------------------------------------------------------------------------------------*\
| VcDevice: Implement the controller event dependend actions for a vacuum chamber device.
|
|
|
|  File: vcdevice.h
|
|  Running level in design of robot controller software: 1                                                      |
|  Task:                                                                                                        |
|            See robotdevice.h; This class is responsible for solving all hardware d ependend operations of the concrete machine|
|            of type "Vacuum Chamber", which consists of 3 valves, one pump, one doc king station (tip rack at biomek 2000),    |
|            one ventilator (part of the drying station) and one heating element (part of the drying station). The d irect      |
|            access is processed with the electronic interface developed and described within the dissertation.|
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                      |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------~*/

#ifndef VC_DEVICE_ROBOT_DEVICE
 #define VC_DEVICE_ROBOT_DEVICE
 #include "robotdevice.h"

 class VcDevice : public RobotDevice {
       public:
                   VcDevice(DevParams stDevParams);       // Passed parameter reports the given hardware dependency of the d e-
vice.
                   ~VcDevice(void);                                  // Controlled destruction of allocated resources.
                   void SpecificOperation(ActionList *);  // Interface: Adaptor structuring pattern. Solved pure virt u-
al::RobotDevice
                   char DeviceReadyForOperation(void);          // Returning internal status of initialization.
       private:
                   DevParams _stDevParams;                           // Structure for managing the hardware parameters of the
device.
```

```
                int _iPort;                                             // File handle parameter for low level output stream
access.
                char _bClassReadyToUse;                         // Flagging internal status of initialization.
 };
#endif
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| VacModel: Implement the controller event dependend actions.
                            |
|
|
|  File: view.h
|
|  Running level in design of robot controller software: 1                                                            |
|  Task:                                                                                                              |
|            See controllerinterface.h; All controller interface dependend calls and callbacks are located here. Norma lly the  |
|            controller will build up an association with the view class to connect a device to it's (time dependend) infor-   |
|            mational system. The view class is a base class to guarantee an inh erited ControllerInterface on one hand and the |
|            proper connection of the explicit device on the other. This solves the implementation of an abstract fact ory- and |
|            strategy pattern.
                                                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_VIEW
 #define ROBOT_VIEW
 #include "controllerinterface.h"
 #include "declarations.h"
class Controller;
class View : public ControllerInterface {
       public:
                   View(void *cptrAnchor);                                        // Receive anchor class and do some initi a-
lizations.
                   ~View(void);                                                   // Controlled destruction of allocated r e-
sources.
       protected:
                   Controller *_cptrAnchor;                                 // Anchor class, which offers an informational i n-
terface.
};


#endif
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Vocabulary: Compiling a vocabulary package.
|                                    |
|
|
|  File: vocabulary.h
|
|  Running level in design of robot controller software: 1                                                            |
|  Task:                                                                                                              |
|            With a given configuration file, the vocabulary for the different actions has to be interpreted. The result is a  |
|            package of an executable vocabulary, which can be used for translation into the language used with the a bstract   |
|            machine. Result of the instance of this class will be an encapsulated package, which is  accessible with a given   |
|            interface function. Together with a compiled grammar package, the langu age system is complete and ready for being |
|            used with the translator instance which will implement a language object (implemented as a mixin class co mposed of|
|            grammar and vocabulary classes).
|                                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                             |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#ifndef ROBOT_VOCABULARY
 #define ROBOT_VOCABULARY
 #include "declarations.h"
class Vocabulary {
public:
            Vocabulary(void);                                                   // Initializations and standardization of
member vars.
            ~Vocabulary(void);                                                  // Controlled destruction of allocated r e-
sources.
            void ReadVocabulary(char *szFile);                                  // Get the vocabulary paragraph from a given conf i-
guration file.
            VocabularyPackage GetVocabulary(void);                      // Interface:Alien objects access the encapsulated vocab u-
lary...                                                                                 // ... package.
            VocabularyPackage GetVocabulary(GrammarPackage);    // With a grammar to be downloaded, the voc package will be...
                                                                                    // ... generated.
private:
            VocabularyPackage _stVocabulary;                            // Encapsulated package of executable vocab ulary.
            void AddToVocList(char *szCmd);                                     // Auxiliary: enlarging a 2 dim vector (part of voc.
package)...
```

```
                                                                                                    // ... by one element.
            int GetElements(char *,GrammarPackage,char *);           // Manager function for element interpretation of an ...
                                                                                                    // ...unknown list.
};


#endif
9.1.2 Source Dateien.
/*~-------------------------------------------------------------------------------------------------------------------*\
| Controller: Managing the abstract machine.                                                                          |
|
|
|  File: controller.c                                                                                                 |
|  Running level in design of robot controller software: 0                                                            |
|  Task:                                                                                                              |
|            Installing a temporary relation to the translator to get the tasks for the concrete machine. Instantiation of the |
|            Views and Model. Definition of an interface (see pure virtuals for ex planation). All in all this class is not pure|
|            virtual to make sure that this controller is not used as an interface d efinition, only. Moreover, the pure inter- |
|            face for that purpose is defined in the class "ControllerInterface" (see file controllerinterface.h for expla-    |
|            nations.)
                                                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "Controller.h"
#include "Translator.h"
#include "Declarations.h"
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables. The command line vector system is copied to corresponding member variables.   |
|            With that, the outer input interface is available by the controllers informational system.
        |
|-------------------------------------------------------------------------------------------------------------------|
```

```
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------------------~*/
Controller::Controller(int iArgs,char **szArgs)
 {
   #ifdef VERBOSE                                                  // Available in demonstration/debug situations, only. ...
       cout << "Initializing Controller...\n";           // ... There will be some output in the standard ...
   #endif                                                                 // ... command window while construction is perfo r-
med.

  _bClassReadyToUse=FALSE;                                         // Private flag, indicating proper initialization of the
instance.
  _szArgs=szArgs;                                                  // Save the argument vector as the only input sou rce...
  _iArgs=iArgs;                                                          // ... and fix it’s size with private member vari a-
bles.
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
             |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
```

```
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being ready for use.
                                |
| On Error:               FALSE is returned if the instance is not properly configured.
               |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char Controller::ClassReadyToUse(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Controller ready to use ?---->";         // ... There will be some output in the standard ...
       cout << _bClassReadyToUse << "\n";
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _bClassReadyToUse;                                         // Private flag, indicating proper initialization of the
instance.
}

/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                     |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
```

```
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
Controller::~Controller(void)
 {
   #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Leaving Controller...\n";                      // ... There will be some output in the standard ...
   #endif                                                                  // ... command window while destruction is perfo r-
med.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member facility as part of the "inner" interface, which allows testing the input vector.
         |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE.
|
| On Error:         FALSE is returned.
                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char Controller::TestInfoLine(void)
 {
```

```
  //... Do some testing here, if informational vector is ok (Syntax, Allocation, etc.).
   #ifdef VERBOSE                                                  // Available in demonstration/debug situations, only. ...
       cout << "Controller: \"TestInfoLine\"\n";       // ... There will be some output in the standard ...
   #endif                                                                  // ... command window while destruction is perfo r-
med.
  return TRUE;
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:       Auxiliary: Excerpt long integer numerical data from ascii streams.
                          |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            The function relieves on a correct formatted string. Therefore, it is important to check that on its reli ability. |
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            In case of an error the negative MAXLONGINT is returned. To ensure the seldom case, that there is really the nega-|
|            tive MAXINT is to be encoded, another error checking system could be i nstalled, later.
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           long integer of the numerical data, being excerped.
                            |
| On error:               MAXLONGINT.
                                                  |
\*-------------------------------------------------------------------------------------------------------------------~*/
long int Controller::GetNumerical(char *szLine,int iPos)
 {
  char bChar;                                                             // Byte for doing some non critical analysis.
  char szBuffer[100];                                                  // Translator string. There shouldn't be a number greater than
100 digits.
  long int lNumber;                                                    // Get the result with this placeholder.
  int iA=0;                                                               // Counter accessing the vector and doing control.
for(;;)                                                                         // As long as there is a vector element.
   {
      bChar=szLine[iPos+iA];                                              // Keep track on offset (iPos) and perform investigation
starting there.
```

```
    if(isdigit(bChar))                                          // If that byte is a number, then take it as a part of the
translator ...
       szBuffer[iA]=bChar;                                      // ... string which will be used to get the prospective
number.
      else                                                             // With an non digit element within the substring,
the archiving ...
       {                                                                  // ... is finished. The translator string is
completed with the ...
        szBuffer[iA]='\0';                                      // ... NULL element and the loop is left for synthesizing
the ...
        break;                                                           // ... long integer immediately after.
    }
      iA++;                                                              // Next element.
      if(iA >= 100)                                                      // Keep track on buffer length.
       return (MAXLONGINT);                                     // On error return some signal (see Improvement in header
section).
   }

  lNumber=atol(szBuffer);                                     // Using system facility to get long integer from string.
  #ifdef VERBOSE                                                     // Available in demonstration/debug situations, only. ...
       cout << "Controller: \"GetNumerical\"\n";       // ... There will be some output in the standard ...
   #endif                                                            // ... command window while destruction is perfor-
med.
  return lNumber;                                                    // Return received long integer to calling function.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Within this function, there will be a relation to a translator instance. The first command line argument is       |
|            passed to the translator. That argument could be a command or a file n ame. If the robot design is e.g. built on   |
|          control-files, there will be the possibility to use scripts or command lists within that files. These could be    |
|            read in and build up rules within the translator instance or language section.
                 |
|            A set of vocabulary is copied from the translator object. This set is needed for YYY
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
```

```
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE.
|
| On Error:         FALSE is returned.
                  |
\*-------------------------------------------------------------------------------------------------------------------------------~*/
int Controller::TranslateArguments(void)
 {
  int iA;                                                                    // Assign elements of current vocabulary.
  char **szBuffer;                                                        // Allocating memory for vocabulary list elements.
  #ifdef VERBOSE                                                          // Available in demonstration/debug situations, only.
    cout << "Controller: \"TranslateArguments...\" \n";
    // cin >> iA;
  #endif
  Translator *cptrTranslator=                                        // Invoking a relation to a translator ...
                             new Translator(_szArgs[1]);             // ... instance. Passing command line elements to the ...
  if(cptrTranslator == NULL) return FALSE;                 // ... translator (e.g. filenames etc.).

  if(!cptrTranslator->ReadyToUse()) return FALSE;   // Using the relation is bound on its proper initialization.
  if(! cptrTranslator->Translate()) return FALSE;   // With correct implementation of the abstract translator, ...
  _stLanguage= cptrTranslator->GetLanguage();             // ... the current language is needed.
  szBuffer=new char *[_stLanguage.stVocabulary._iNumVocListElements]; // The co ntroller needs a set of vocabulary.
  if(szBuffer == NULL) return FALSE;                      // With unsuccessful allocation, the controller should bail out.

  for(iA=0;iA < _stLanguage.stVocabulary._iNumVocListElements; iA++)  // Copying the set of vocabulary.
   {
    szBuffer[iA]= new char [strlen (_stLanguage.stVocabulary._szVocList[iA])+1];
    strcpy(szBuffer[iA],_stLanguage.stVocabulary._szVocList[iA]);
   }
  _stLanguage.stVocabulary._szVocList= szBuffer;    // Storing the set of vocabulary in the LanguagePackage struct ure.
  #ifdef VERBOSE                                                          // Available in demonstration/debug situations, only.
    cout << "Controller: Specific language transferred to robot’s world. \n";
    // cin >> iA;
  #endif
  delete cptrTranslator;                                             // Unlink from translator-relation.
  _bClassReadyToUse=TRUE;                                            // Flagging proper initialization.
  return TRUE;
 }

/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task:      Internal interface: Return depth of argument vector stored within this class.
             |
```

```
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                 |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value: Depth of argument vector.
                               |
\*-------------------------------------------------------------------------------------------------------------------------~*/
int Controller::GetAmountArguments(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Controller: Get depth args.\n";          // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _iArgs;                                                    // Private member: Depth of argument vecto r stored with **
_szArgs.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Internal interface: Return argument vector stored within this class.
                    |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
```

```
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value: Adress of argument vector.
                               |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char **Controller::GetArgumentVector(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Controller: Get argument vector.\n";  // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _szArgs;                                                   // Private member: Argument vector depth stored in _iArgs.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Internal interface: Return struct containing compiled Grammar and Vocabulary.
             |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value: Language package structure.
                               |
```

```
\*-------------------------------------------------------------------------------------------------------------------------~*/
LanguagePackage Controller::GetLanguagePackage(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Controller:\"GetLanguagePackage\"\n"; // ... There will be some output in the standard ...
   #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _stLanguage;                                               // Struct containing compiled Grammar and Vocabulary.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Internal interface: Return struct containing a list of device events.
                    |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value: Device events structure.
                               |
\*-------------------------------------------------------------------------------------------------------------------------~*/
TaskSchedule Controller::GetTaskPackage(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Controller:\"GetTaskPackage\"\n";      // ... There will be some ou tput in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _stTaskSchedule;                                                // Struct containing a list of device events.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Internal interface: Return struct containing action list.
                             |
```

```
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value: Device action structure.
                               |
\*-------------------------------------------------------------------------------------------------------------------------~*/
ActionList *Controller::GetCurrentAction(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Controller:\"GetCurrentAction\"\n";       // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _stActionList;                                             // Struct containing a list of device actions.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| ControllerInterface: Entail an interface for control purposes.                                                            |
|
|
|  File: controllerinterface.c                                                                                              |
|  Running level in design of robot controller software: 0                                                                  |
|  Task:                                                                                                                    |
|            The model class and the view class should be accessible with the same i nterface. With that strategy, there is an  |
|            easy way to exchange descendants within the controller. Advantage with Patterns like "Abstract Factories" and     |
|            "Strategy Patterns". The interface is pure virtual, therefore. It’s task is to entail an interface.
         |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
```

```
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-----------------------------------------------------------------------------------------------------------------~*/
#include <iostream.h>
#include "ControllerInterface.h"
#include "Controller.h"
/*~-----------------------------------------------------------------------------------------------------------------*\
| Task: Initialization and implementation of a pure (abstract) interface class.
                     |
|-----------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-----------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-----------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-----------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                    |
|-----------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE.
|
| On Error:        FALSE is returned.
                   |
\*-----------------------------------------------------------------------------------------------------------------~*/
ControllerInterface::ControllerInterface(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Initializing ControllerInterface.";       // ... There will be some output in the standard ...
       cout << "(being abstract)\n";
```

```
  #endif                                                                  // ... command window while destruction is perfo r-
med.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Controlled destruction of this pure (abstract) interf ace class.
                          |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                 |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE.
|
| On Error:        FALSE is returned.
                   |
\*-------------------------------------------------------------------------------------------------------------------~*/
ControllerInterface::~ControllerInterface(void)
 {
  #ifdef VERBOSE                                                  // Available in demonstration/debug situations, only. ...
      cout << "Leaving ControllerInterface";            // ... There will be some ou tput in the standard ...
      cout << "(being abstract)\n";
  #endif                                                                  // ... command window while destruction is perfo r-
med.
 }

/*~-------------------------------------------------------------------------------------------------------------------*\
| Grammar: Compiling a grammar package.
                                          |
|
|
```

```
|  File: grammar.c
|
|  Running level in design of robot controller software: 0                                                                    |
|  Task:                                                                                                                      |
|            Compiling a grammar list which contains the rules with which commands wit hin a configuration file will set up the |
|            corresponding action list of an abstract machine. Finally this grammar will be used together with a given vocabu- |
|            lary to compose a language which can be used to translate the given tasks into the abstract tasklist of the ab-   |
|            stract machine.    A given configuration file, which contains the grammar to be used, will be investigated. The exe-
|
|            cutable grammar elements will be compiled to a final package of encaps ulated grammar rules (GrammarPackage).    |
|            This result will be accessible with an interface function, then.
                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                                    |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include "Declarations.h"
#include "Grammar.h"
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Initializations and standardization of member vars.
                                    |
|-----------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-----------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-----------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                    |
\*-----------------------------------------------------------------------------------------------------------------------~*/
Grammar::Grammar(void)
 {
  #ifdef VERBOSE                                                          // Available in demonstration/debug situations, only. ...
       cout << "Initializing Grammar...\n";        // ... There will be some output in the standard ...
  #endif                                                                        // ... command window while destruction is perfo r-
med.
  _szCmdList=NULL;                                                      // 2 dimensional list of executable grammar elements. These el e-
ments ...
  _iNumCmdListElements=0;                                     // ... will be identified and compiled to a final package of grammar
rules.
  _stGrammar._iDevices=0;                                       // Encapsulating compiled rules. Performing initializations of
...
  _stGrammar._bDeviceSymbol=NULL;                               // ... member variables to be listed, there.
  _stGrammar._bTimeSymbol=0;
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
             |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
```

```
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                    |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being ready for use.
                                  |
| On Error:               FALSE is returned if the instance is not properly configured.
                |
\*-------------------------------------------------------------------------------------------------------------------------------~*/
char Grammar::ClassReadyToUse(void)
 {
  #ifdef VERBOSE                                                        // Available in demonstration/debug situations, only. ...
        cout << "Grammar ready to use ?---->";            // ... There will be some output in the standard ...
        cout << _bClassReadyToUse << "\n";
  #endif                                                                      // ... command window while destruction is perfo r-
med.
  return _bClassReadyToUse;                                             // Private flag, indicating proper initialization of the
instance.
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task: As part of the public interface, this function returns the encapsulated package of compiled rules.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                    |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           Package of compiled rules.
                                           |
\*-------------------------------------------------------------------------------------------------------------------------------~*/
GrammarPackage Grammar::GetGrammar(void)
 {
```

```
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
   cout << "Grammar:Returning grammar package...\n";// ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  return _stGrammar;
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      During compilation of a grammar package, there is the need of enla rging a 2 dimensional package of complete       |
|            command lists. This internal function does the enlarging process and is protected, therefore. The facility could  |
|            be useful for siblings and may be inherited.
                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                          |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void Grammar::AddToCmdList(char *szCmd)
 {
  char **szNewCmdList;                                              // Offering a new anchor for an enlarged command list ve c-
tor.

  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
   static int siCounter=1;                                          // ...
   cout << "Grammar: Adding " << siCounter << "nd"; // ... There will be some output in the standard ...
   cout << " informational package.\n";                          // ...
   siCounter++;                                                           // ... command window while destruction is perfo r-
med.
  #endif
```

```
  szNewCmdList= new char *[_iNumCmdListElements+1]; // To introduce a new command list element, the memory is to be ...
  if(szNewCmdList == NULL)                                        // ... allocated. The allocation is to be controlled as
usual.
   return;
  for(int iA=0;iA < _iNumCmdListElements; iA++)           // The 2 dimensional vector will first safe the elder data ...
    szNewCmdList[iA]=_szCmdList[iA];                      // ... using simple adress transfer.
  delete [] _szCmdList;                                           // Old anchor is not needed anymore.#
  _szCmdList=szNewCmdList;                                        // Using the new anchor will enable to do a adress tran s-
fer...
  _szCmdList[_iNumCmdListElements]=szCmd;                 // ... to hook on the next command package ...
  _iNumCmdListElements++;                                       // ... and the package vector will appear as being enlarged.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Download the given rules to a structure managing the grammar. The heart of that facility is to recognize sub-     |
|            strings within a vector of given command strings. These substrings are used as indicators for the following data. |
|            goal is to fix two quenstions:
                                          |
|                                                                 1. Is something given which can be used to build a gra m-
mar.                                |
|                                                                 2. If there is a grammar element, what is it coding for.
                              |
|
                                                                |
|            Example:
                                                                |
|            Mainly (with foundation at 15.03.1996 by GKA) fo llowing indicators were used for the vacuum chamber system:
|
|
                                                                |
|                          GRAMMAR_SEPARATOR_INDICATOR                    := "SEP"
                                        |
|                          GRAMMAR_TIME_INDICATOR                         := "TIM"
                                        |
|                          GRAMMAR_CYCLING_INDICATOR                 := "CYC"
                                   |
|                          GRAMMAR_PARENTHESIS_INDICATOR             := "PAR"
                                   |
|                          GRAMMAR_ENUMERATOR_INDICATOR              := "ENU"
                                   |
|                          GRAMMAR_DEVICE_INDICATOR                  := "DEV"
                                   |
|                          GRAMMAR_COMMENT_INDICATOR                 := "COM"
                                   |
|
                                                                |
|            With increasing the flexibility by use of a profiling configuration file system (see section "Improvement" within |
```

```
|            this header) these indicators may become obsolete.
|
|
|            With these original indicators it is possible to identify the foll owing important mnemonics to build a grammar:
|
|                        1. Separate one element from another (mainly "-")
|
|                        2. Time indicators for indicating duration s, actions happening at defined time scale, etc. Mainly a
"t".  |
|                        3. Loops are needed for recoursive actions (e.g. cyclic opening of ventiles at defined time intervals).
|
|                           Mainly a "c".
|
|                        4. Indicating periodic or enclosed or grouped events. The actions or events will be grouped with a co u-
ple |
|                           of elements, always: An opening parenthesis element (mainly "(") and a closing parenthesis element
|
|                           (mainly ")").
|
|                        5. Enumeration in all cases (mainly ",").
|
|                        6. Decoding of devices (e.g. "V" for Valve or "P" for Pump).
|
|                        7. To enable comments within the profiling file system (do not mix this expression with the mentioned
des-|
|                           cription of a more flexible use of a profiling !configuration! file system) there is need of some
sym- |
|                           bols indicating comments (mainly "//" like C++ comments).
|
|
|            Example of a profiling file system used with foundation level (15.03.1996 by GKA):
|
|
|            <Start of file with the next line in this text:>
|                           GRAMMAR:
|                           SEP:-
|                           TIM:t
|                           CYC:c
```

```
|                              PAR:()
|                                                    |
|                              ENU:,
|                                                    |
|                              DEV:V
|                                                    |
|                              DEV:P
|                                                    |
|                              DEV:O
|                                                    |
|                              COM://
|                                                    |
|                              ACTIONS:
|                                                    |
|                              -V1t5,125          // Open valve 1 after 5 seconds for 125 seconds.
|                          |
|                              -V3tc(5,1)         // Open valve 3 all 5 seconds for 1 second.
|                                |
|                              -Pt0               // Run pump with second 0 (immediately).
|                                |
|                              -Ot20              // Robot terminates with 20 seconds of run.
|                                |
|           <End of file with the line before this text.>
|                                        |
|
|                                                                 |
|
|                                                                 |
|           This is a complete file. As one can see, there are basically two sections. The first is the grammar sect ion,      |
|           starting with the mnemonic GRAMMAR: with all rules mentioned above.The other is the ACTION section, decoded later.|
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|           none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|           inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|           It is possible to enlarge the flexibility of the complete grammar system with using editable configuration files  |
|           containing the mnemonical GRAMMAR_XXX_INDICATORs and solving them by reading them during object initializ ation    |
|           within the constructor. These indicators won't be used by alien classes. Therefore it would be a good d esign deci-|
|           sion to use a dynamically allocated structure containing some string variables, which then solve the ind ication.  |
```

```
|            With leaving the Grammar object, all that data could be freed, then.
                    |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                   |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void Grammar::GenerateGrammar(void)
 {
  char *bDeviceSymbol=NULL;                                          // It is not known, how much devices will be used with
that "virtual ...
  char *bTimeSymbols=NULL;                                           // ... robot". Could be, that this is a conglomerate of
some ...
                                                                            // ... machines. Therefore a dynamically a l-
located vector system, ....
                                                                            // ... which can enlarge during following
decoding is needed.

  for(int iA=0 ; iA < _iNumCmdListElements; iA++)   // Analyzing the complete GRAMMAR: package (source= profiling ...
   {                                                                       // file system). Explanations see Task section in
header above.
      if(strstr(_szCmdList[iA],GRAMMAR_SEPARATOR_INDICATOR) != NULL)                        // Doing compilation of ...
       _stGrammar._bSeparator= _szCmdList[iA][GRAMMAR_IDENTPOSITION];                       // ... grammar elements ...

      // ... used with the current...

      // ... machine. With this ...
      if(strstr(_szCmdList[iA],GRAMMAR_CYCLING_I NDICATOR) != NULL)                        // ... steps, the profiling ...
       _stGrammar._bCycleSymbol= _szCmdList[iA][GRAMMAR_IDENTPOSITION];                    // ... file system defines ...
      if(strstr(_szCmdList[iA],GRAMMAR_PARENTHESIS_INDICATOR) != NULL)                 // ... the machine and the ...
       {
      // ... grammar understood by ...
        _stGrammar._bOpenParenthesis= _szCmdList[iA][GRAMMAR_IDENTPOSITION];           // ... it’s action definitions, ...
        _stGrammar._bCloseParenthesis= _szCmdList[iA][GRAMMAR_IDENTPOSITION+1];        // ... written in the "language" ...
       }
      // ... only this robot "under- ...
      if(strstr(_szCmdList[iA],GRAMMAR_ENUMERATOR_INDICATOR) != NULL)                       // ... stands". As an explanati-
on ...
       _stGrammar._bEnumerator= _szCmdList[iA][GRAMMAR_IDENTPOSITION];                  // ... see the discussion in Task ...

      // ... section, above and compare ...
    if(strstr(_szCmdList[iA],GRAMMAR_DEVICE_INDICATOR) != NULL)                         // ... it to the "vacuum chamber
...
```

```
       {
      // ... language" used with the ...
        bDeviceSymbol= new char [_stGrammar._iDevices+1];                                   // ... ACTIONS: section
within the ...
        for(int iB=0;iB < _stGrammar._iDevices; iB++)                                          // ... same file. No
other "robot" ...
         bDeviceSymbol[iB]=_stGrammar._bDeviceSymbol[iB];                                   // ... may understand "V"
for valve.
        bDeviceSymbol[_stGrammar._iDevices]=_szCmdList[iA][GRAMMAR_IDENTPOSITION];
        delete [] _stGrammar._bDeviceSymbol;
        _stGrammar._bDeviceSymbol= bDeviceSymbol;
      _stGrammar._iDevices++;
       }
      if(strstr(_szCmdList[iA],GRAMMAR_TIME_INDICATOR) != NULL)
       _stGrammar._bTimeSymbol= _szCmdList[iA][GRAMMAR_IDENTPOSITION];
      if(strstr(_szCmdList[iA],GRAMMAR_COMMENT_INDICATOR) != NULL)
       {
        _stGrammar._szCommentSymbol[0]= _szCmdList[iA][GRAMMAR_IDENTPOSITION];
        _stGrammar._szCommentSymbol[1]= _szCmdList[iA][GRAMMAR_IDENTPOSITION+1];
        _stGrammar._szCommentSymbol[2]= '\0';
       }
   }
  #ifdef VERBOSE
    cout << "Grammar: "<< _stGrammar._iDevices << " Devices found\n";
       cout << "Grammar: Separator  symbol= "<< _stGrammar._bSeparator << "\n";
    cout << "Grammar: "<< _stGrammar._bTimeSymbol << " Time Symbol found\n";
       cout << "Grammar: Cycling symbol= "<< _stGrammar._bCycleSymbol << "\n";
       cout << "Grammar: Parenthesis open= "<< _stGrammar._bOpenParenthesis << "\n";
    cout << "Grammar: Parenthesis close= "<< _stGrammar._bCloseParenthesis << "\n";
    cout << "Grammar: Enumerator= "<< _stGrammar._bEnumerator << "\n";
       for(iA=0;iA< _stGrammar._iDevices;iA++)
        cout << "Grammar: Device symbol(" << iA << ")= " << _stGrammar._bDeviceSymbol[iA] << "\n";
       cout << "Grammar: Comment symbol= "<< _stGrammar._szCommentSymbol << "\n";
  #endif

 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:       Get the grammar paragraph from a given configuration file. The gra mmar of a robot's language is stored within a  |
|           section of the profiling file system. With the foundation (at 15.03.1996 by GKA) there is one file conta ining the|
|             PROFILING_FILE_SYSTEM_GRAMMAR_INDICATOR "GRAMMAR:" and the PROF ILING_FILE_SYSTEM_ACTION_INDICATOR "ACTIONS:".    |
|             Within a grammar object, the grammar section is investigated, only. It's elements will be analyzed (within the   |
|             member function GenerateGrammar(). To get the complete section, the grammar section of profiling file is read in |
|             here.
                                                    |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
```

```
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            It is possible to enlarge the flexibility of the complete grammar s ystem with using editable configuration files  |
|            containing the mnemonical PROFILING_FILE_SYSTEM_XXX_INDICATORs and solving them by reading them during o bject     |
|            initialization within the constructor. These indicators won't be used by alien classes. Therefore it would be a   |
|            good design decision to use a dynamically allocated s tructure containing some string variables, which then solve  |
|            the indication. With leaving the Grammar object, all that data could be freed, then.
|              |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
|                   |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|                                                    |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void Grammar::ReadGrammarList(char *szFile)
 {
  char *szBuf;                                                                  // The profiling system files are small enough to
fit in one buffer.
  char *szNewCmd;                                                             // Each informational package will be fed into this bu f-
fer. The ...
                                                                                      // AddToCmdList() function will reset these
elements into a 2Dim ...
                                                                                      // ... vector: **_szCmdList. This member v a-
riable will then be  ...
                                                                                      // ... used to compile a final package of
grammar rules.
  char bGrammar=FALSE;                                                        // Flagging whether there is a proper profiling system or
not.

  ifstream smCfgFile;                                                         // Doing some file analysis, first. Because the files used
for ...
  smCfgFile.open(szFile,ios::in);                                       // ... profiling are small enough to fit in one byte str eam, ...
  if(!smCfgFile)                                                              // ... the file size is to be defined.
   cout << szFile << " is missing.";

  smCfgFile.seekg(0,ios::end);                                          // Reset to end of file
```

```
  int iFileSize=smCfgFile.tellg();                                  // Get the file size.
  smCfgFile.seekg(0,ios::beg);                                      // Reset to the start of the file.
  #ifdef VERBOSE
   cout << "Grammar: Reading grammar section of profiling file system.\n";
   cout << "         Using <" << szFile << ">  (" << iFileSize << ") Bytes\n";
  #endif
  if(iFileSize > MAX_CFGFILESIZE) return;                 // Being sure, file is fitting to predefined settings.
  szBuf= new char[iFileSize+1];                                     // Doing controlled dynamic allocations ...
  if(szBuf == NULL) return;                                               // ... .


  while(smCfgFile)                                                  // Because the PROFILING_FILE_SYSTEM_GRAMMAR_INDICATOR ...
   {                                                                      // ... which is e.g. "GRAMMAR:" may be anywhere in
the ...
    smCfgFile.getline(szBuf,iFileSize,'\n');       // ... file, the complete stream is possibly to be in- ...
                                                                                  // ... vestigated.
    if(!strcmp(PROFILING_FILE_SYSTEM_GRAMMAR_INDICATOR,szBuf))   // If the indicator is found, start ...
       bGrammar=TRUE;                                                                 // ... investigation (see
beyond).
      if(!strcmp(PROFILING_FILE_SYSTEM_ACTION_INDICATOR,szBuf))   // If "GRAMMAR:" is before "ACTIONS:", the ...
       bGrammar=FALSE;                                                          // ... reading of grammar rules is in-
terrupted ...
                                                                                                 // ... immediately.
      if(bGrammar)                                                  // Starting investigation, after indicator is being found.
       {
        if(bGrammar > TRUE)                                         // Excerpt the rules containing lines, only. Be sure, tha t
...
         {                                                                    // ... there will be an allocation with the first
line ...
            szNewCmd= new char [strlen(szBuf)+1];        // ... (!) within (!) the "GRAMMAR:" section, only. The ...
            strcpy(szNewCmd,szBuf);                                 // flag bGrammar is on TRUE (=1) in case of finding
"GRAMMAR:".
          AddToCmdList(szNewCmd);                                   // Feed grammar element into the 2Dim vector **_szCmdList,
there.
         }
        else
         bGrammar++;                                                          // A new allocation is needed.
       }
   }
  delete [] szBuf;                                                  // Freing resources.
 }


/*~-------------------------------------------------------------------------------------------------------------*\
| Task: If there are allocated resources, the destructor will free them. Because all compiled grammar is kept in these    |
|           dynamic allocations, the virtual robot looses it's ability to understand the language, used. This destru ctor      |
```

```
|            should therefore only be used after all specific grammar elements are transferred to the robot's world.
|
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                    |
\*-----------------------------------------------------------------------------------------------------------------~*/
Grammar::~Grammar(void)
 {
  #ifdef VERBOSE
   cout << "Grammar: On destruction.\n";
  #endif
  if(_szCmdList != NULL)                                        // If there is an informational packa ge, ...
   {                                                                      // ... destroy it's elements and ...
       for(int iA=0;iA< _iNumCmdListElements;iA++)         // ... free the associated ...
        delete [] _szCmdList[iA];                                       // ... 2Dim vector, too. ...
       delete [] _szCmdList;                                            // ... .
   }
  if(_stGrammar._bDeviceSymbol != NULL)                         // If there is a device-symbol, ...
   delete [] _stGrammar._bDeviceSymbol;                         // ... destroy it.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Language: Mixin class, unifying the two members of a language:  grammar and vocabulary.
        |
|
|
|  File: language.c
                                                    |
```

```
|  Running level in design of robot controller software: 2                                                       |
|  Task:                                                                                                         |
|            Unifying grammar and vocabulary of an abstract language into one accessible language package.
         |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                       |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*--------------------------------------------------------------------------------------------------------------~*/
#include <stdio.h>
#include <iostream.h>
#include "Declarations.h"
#include "Language.h"
/*~--------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.  Language is meant to be a mixin-class of Grammar and Vocabulary.  |
|---------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|---------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|---------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|---------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                   |
|---------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*--------------------------------------------------------------------------------------------------------------~*/
Language::Language(void)
```

```
 {
  #ifdef VERBOSE
   cout << "Initializing Language.\n";
  #endif
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Interface: Unifying vocabulary and grammar into one language package. Make this encapsulated package accessible   |
|            to aliens.
                                                      |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:    LanguagePackage.
                                 |
\*-------------------------------------------------------------------------------------------------------------------~*/
LanguagePackage Language::GetLanguage(void)
 {
  _stLanguage.stGrammar=GetGrammar();                    // Master task of this mix in class is to compile the la nguage ...
  _stLanguage.stVocabulary=GetVocabulary();              // ... from an existing grammar and an existing vocabulary.
  #ifdef VERBOSE
   cout << "Language: Returning language\n";
  #endif
  return _stLanguage;                                                  // With having grammar and vocabulary, the language is
ready to use.
 }

/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: If available, freing allocated resources.   Language is meant to be a mixin-class of Grammar and Vocabulary.   |
|-------------------------------------------------------------------------------------------------------------------|
```

```
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------------------~*/
Language::~Language(void)
 {
  #ifdef VERBOSE
       cout << "Leaving Language...\n";
  #endif
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Controller: Main entry point of using the abstract machine.                                                                    |
|
|
|  File: main.c                                                                                                                  |
|  Task:                                                                                                                         |
|            Offering a sample application (model system in this case) to demonstrate the use of the virtual robot library     |
|            system.
                                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                                       |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
```

```
|                  1.0
|
\*-------------------------------------------------------------------------------------------------------------------------~*/

#include <iostream.h>
#include "Declarations.h"
#include "VacuumChamber.h"
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task: Sample function for starting a robot simulation. In this case the vacuum cha mber controller is used for this pur- |
|            pose, because it’s funcionality is predestinated for a terminal printout simulation.
              |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                   |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    on error   FALSE
                                |
|                         on success TRUE
                                                          |
\*-------------------------------------------------------------------------------------------------------------------------~*/
int RunRobot(int iArgs, char **szArgs)
 {
  cout << "################################## Make new VacuumChamber ######################\n";
  VacuumChamber *cptrController=new VacuumChamber(iArgs,szArgs); // Initiate a new controller of type va cuum chamber.

  if(!cptrController->TranslateArguments())                          // Synthesize grammar, vocabulatory and get task scedule.
   {
      delete cptrController;                                                  // Having troubles there, does not allow
proceeding.
    return FALSE;
```

```
   }

  cout << "################################## Run VacuumChamber ##############################\n";

  cptrController->Run();                                                  // Having all tasks within the controlles' virtual
world,
                                                                                  // ... start running the controller,
  delete cptrController;                                                  // With finishing, destroy the virtual robot.
  return TRUE;
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task: Main entry point of the application.
                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    on error   ErrorCode
                                  |
|                         on success Zero
                                                        |
\*-------------------------------------------------------------------------------------------------------------------------~*/
int main(int iArgs, char **szArgs)
 {
  if(iArgs < 2)
   {
    cout << "Configuration file missing.\n";
       return 1; }
  cout << "################################## Main function ##################################\n";
  cout << "Running virtual robot with configuration file << " << szArgs[1] << "\n";
```

```
  if(!RunRobot(iArgs,szArgs))
   return 1;
  return 0;
 }
/*~------------------------------------------------------------------------------------------------------------------*\
| Model: Implementation of a standard model, which can be inherited to the concrete machines.
        |
|
|
|  File: model.c
                                                    |
|  Running level in design of robot controller software: 1                                                           |
|  Task:                                                                                                             |
|            The model class is a base class for concrete machine (dependend) m odels. It excerps all the parts which may be    |
|            found in all models and therefore stand per definitionem for the expression of "model".    The anchor class is sol-
ved|
|            here and the abstract task list is managed, too.
                                 |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                           |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*------------------------------------------------------------------------------------------------------------------~*/
#include "Model.h"
#include "Controller.h"
#include <iostream.h>
/*~------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables. Storing of an anchor class, providing an informational interface.               |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
```

```
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, inven ted by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
Model::Model(void *cptrAnchor) : ControllerInterface()
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Initializing Model (abstract base class)\n";    // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
  _slCycleInterruptor=NULL;                                         // Initialize cycle counter, needed for cyclic actions
like V3tc(5,1).
  _cptrAnchor= (Controller *) cptrAnchor;                // Anchor class, which offers an informational interface.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task: If available, freeing all allocated resources.
                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
```

```
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------~*/
Model::~Model(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Leaving Model (abstract base class)\n";         // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Robot: Implement the controller event dependend actions for an abstract robot device.
              |
|
|
|  File: robot.c                                                                                                      |
|  Running level in design of robot controller software: 2                                                            |
|  Task:                                                                                                              |
|            See view.h; To solve hardware-dependency, the abstract robot will get a ccess to a concrete machine (RobotDevice)  |
|            which is an object of it’s own.
                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#include <iostream.h>
#include "Robot.h"
#include "Controller.h"
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.
                                       |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
```

```
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
Robot::Robot(void *cptrAnchor,RobotDevice *cptrMachine) : View(cptrAnchor)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Initializing Robot...\n";                    // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while construction is perfo r-
med.
  _bClassReadyToUse=FALSE;                                          // This flag indicates the internal initialization status.
  if(cptrMachine==FALSE) return;                          // The machine-class is needed to solve all machine dependend operat i-
ons ...
  _cptrMachine=cptrMachine;                 // ... of the virtual robot. If there is’nt any, ther is no use with further proce e-
dings.
  _bClassReadyToUse=TRUE;                                           // Initialization complete. Alien classes may continue with this
object.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
             |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being ready for use.
                                  |
| On Error:               FALSE is returned if the instance is not properly configured.
               |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char Robot::ClassReadyToUse(void)
 {
  return _bClassReadyToUse;                                          // Private flag, indicating pro per initialization of the
instance.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member. It is responsible for solving the pure virt ual::ControllerInterface functionality. This, to be    |
|            consistent with that pure abstract interface class. With current d esign, the Model and View class (View is parent |
|            of Robot) share the same interface class, which eases transfer operability with the Controller class.
|
|            To implement it's functionality, an action list of the current situation is ordered, first. With that act ion list,|
|            the operation needed to be executed is solved within the Device associ ation (_cptrMachine->specificOperation),    |
|            later in the context.
                                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
```

```
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                  |
\*-----------------------------------------------------------------------------------------------------------------------------~*/
void Robot::Synchronize(void)
 {
  ActionList *stActionList;                                          // A composition of amount and list of device associated
actions.
  stActionList= (ActionList *)                                     // With this list, ...
                        _cptrAnchor->GetCurrentAction(); // ... the currently needed action of the virtual robot is documented.
  #ifdef VERBOSE
   cout << "Robot: synchronizes active devices\n\twith getting the current abstract action and\n";
  #endif
  Operation(stActionList);                                           // Invoking the operation of the concrete device with an
association.
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member. It uses an association with the concrete device to solve the current operation of the vi rtual rob.|
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                 |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                  |
\*-----------------------------------------------------------------------------------------------------------------------------~*/
void Robot::Operation(ActionList *stOnTaskList)
 {
  #ifdef VERBOSE
   cout << "\tsolving it with the associated specific operation,\n\twhich is now:\n\t\t";
  #endif
```

```
  _cptrMachine->SpecificOperation(stOnTaskList);          // Association with the concrete machine.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. It is responsible for solving the pure virt ual::ControllerInterface functionality. This, |
|            to be consistent with that pure abstract interface class. With current design, the Model and View class (View is  |
|            parent of Robot) share the same interface class, which eases transfer operability with the Controller class.    |
|            Alien classes may invoke an update of the current situation with this function. To perform this, the alien does   |
|            not keep track on the current situation of the robot. An invalidation causes the virtual robot system to check    |
|            it's current action database and react on the needs documented, there.
                    |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                  |
\*-------------------------------------------------------------------------------------------------------------------~*/
void Robot::Invalidate(void)
 {
  cout << "Robot invalidated\n";
  Synchronize();                                                  // Performing the action needed currently.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. It is responsible for solving the pure v irtual::ControllerInterface functionality. This, |
|            to be consistent with that pure abstract interface class. With current design, the Model and View class (View is  |
|            parent of Robot) share the same interface class, which eases transfer operability with the Controller class.    |
|            Alien classes may invoke an update of the current situation with this function. To perform this, the alien does   |
|            not keep track on the current situation of the robot. An invalidation causes the virtual robot system to  check    |
|            it's current action database and react on the needs documented, there. This member allows passing a long parameter|
```

```
|            for class internal use (if needed).
                                                 |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                      |
\*-----------------------------------------------------------------------------------------------------------------------------~*/
void Robot::Invalidate(long int lUnit)
 {
  cout << lUnit << ": Robot invalidated\n";              // If needed, a long parameter may be passed ...
  _lCurrentUnit=lUnit;                                             // ... and stored internally.
  Synchronize();                                                   // Performing the action needed currently.
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                     |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
```

```
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
Robot::~Robot(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only.  ...
       cout << "Destroying Robot...\n";                          // ... There will be some output in the standard ...
  #endif                                                                 // ... command window while destruction is perfo r-
med.
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| RobotDevice: Implement the controller event dependend actions for a concrete robot device.
|
|
|
|  File: robotdevice.c
|
|  Running level in design of robot controller software: 0                                                                 |
|  Task:                                                                                                                   |
|            The concrete device is implemented with a strategy method, so that different concrete robots may be excha nged     |
|            during application’s runtime. Therefore all devices should offer  a unique interface for their access. This inter- |
|            face is inherited from this base class. To avoid artificial enlargement of inheritance in the overall d esign, that|
|            class is not meant to be abstract. If there is need to fix some uniquely usable routines here, one may do so. The |
|            main task of this class is to entail an interface to all concrete mach ines. All further development touching the  |
|            interface-behaviour of the concrete machine should use this class for that purpose.
              |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                                 |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
```

```
\*-------------------------------------------------------------------------------------------------------------------------~*/
#include "RobotDevice.h"
#include <iostream.h>
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.  The robot device object is the abstract interface class of the  hardware      |
|            dependent device solution. It serves as an interface class to support a common association to the abstract robot  |
|            class which is responsible for synchronizing the current actions (documented in the action list) to be executed   |
|            in time with the hardware solving functionality "specific operation" which is defined as pure virtual wihtin this |
|            class. That point stresses out, that this class is meant to be an implementation of an abstract interface object, |
|            only, even if it is not per definitionem (RobotDeviceRunning is not pure virtual).
|              |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
|                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
RobotDevice::RobotDevice(DevParams stParams)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Initializing RobotDevice...";                 // ... There will be some output in the standard ...
  #endif                                                                 // ... command window while construction is perfor-
med.
  _bOmegaSystemRunning=TRUE;                                    // The device-conglomerate is up and running at this moment. ...
                                                                         // ... If there will be an Omega-signal
(Otxx), this device terminates.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
```

```
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                     |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
RobotDevice::~RobotDevice(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Terminating RobotDevice.\n";             // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Returning if the robot device is running or terminated. Used with alien query.
        |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
```

```
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being running.
                                        |
| On Error:               FALSE is returned if the robot device is terminated.
                    |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char RobotDevice::RobotDeviceRunning(void)
 {
  return _bOmegaSystemRunning;                                    // Flag indicating the current status.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Translator: Generate language for abstract machine.
                                  |
|
|
|  File: translator.c
                                                    |
|  Running level in design of robot controller software: 0                                                                  |
|  Task:                                                                                                                    |
|            Generating a list of grammar and vocabulary, which is ready for use with the abstract machine.
|
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                                  |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include "Translator.h"
#include "Declarations.h"
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.  The robot device object is the abstract interface class of the  har dware      |
|            dependent device solution. It serves a s an interface class to support a common association to the abstract robot  |
|            class which is responsible for synchronizing the current actions (documented in the action list) to be executed   |
|            in time with the hardware solving functionality "specific operation" which is defined as pure virtual wihtin this |
|            class. That point stresses out, that this class is meant to be an impl ementation of an abstract interface object, |
|            only, even if it is not per definitionem (RobotDeviceRunning is not pure v irtual).
|              |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
|                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------~*/
Translator::Translator(char *szActionFile)
 {
  #ifdef VERBOSE                                                    // Available in demonstrat ion/debug situations, only. ...
      cout << "Initializing Translator.\n";             // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while construction is perfo r-
med.
  _bClassReadyToUse=FALSE;                                          // Flag indicates current initializing status of o bject.
  _cptrLanguage=NULL;                                               // The translator class will have an association with a
Language object.
  _szActionFile=NULL;                                               // Configuration file (source: command line vector [1]) of
current ...                                                                    // ...device.
  _szActionFile = new char [strlen(szActionFile)+1];// The configuration file co ntains two major parts. The first is
  if(_szActionFile == NULL) return;                                 // ... the grammar and vocabulary, used. The other is submitted
...
  strcpy(_szActionFile,szActionFile);                    // ... within the languag e (ACTION-Field) and documents the actions ...
```

```
                                                                                                  // ... which should be executed with the de-
vice conglomerate ...                                                                                                          // ...
"RobotDevice".
  _cptrLanguage= new Language();                              // The translator has an association with a language object, which ...
  if(_cptrLanguage == NULL) return;                                 // ... is a real association: Lifetime as long as the translator
exists.
  _bClassReadyToUse=TRUE;                                           // All initializations went fine. Object ready.
  _stLanguage.bStructValid=FALSE;                                   // Nothing fed into the language package.
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Alien classes may cause a translation of instructions within the configuration file into |
|            the abstract action list of the virtual robot.
                                        |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                        |
| On Error:               FALSE is returned on error.
                                        |
\*-------------------------------------------------------------------------------------------------------------------------------~*/
char Translator::Translate(void)
 {
  if(_szActionFile == NULL)                       // If there are no instructions given, there is no use ...
      return FALSE;                                     // ... to run a translation.
  GetActionList();                                // Translate instructions into a valid action list for the virtual robot sy-
stem.
  return TRUE;
 }
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Alien classes may get the current language package from the translator.
|
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           LanguagePackage in case of success.
                                    |
| On Error:               NULL is returned on error.
                                    |
\*-------------------------------------------------------------------------------------------------------------------~*/
LanguagePackage Translator::GetLanguage(void)
 {
  if(_cptrLanguage != NULL)                                                    // If there is a valid language object ...
   {                                                                                  // ... which contains the services for
building ...
    _stLanguage=_cptrLanguage->GetLanguage();                          // ... an abstract language package.
      _stLanguage.bStructValid=TRUE;                                             // The language package consists of a grammar pack a-
ge and...
   }
  else _stLanguage.bStructValid=FALSE;                                 // ... a vocabulary package.
  return _stLanguage;
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member function.|
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
```

```
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                    |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void Translator::GetActionList(void)
 {
  _cptrLanguage->ReadGrammarList(_szActionFile);                // Get the grammar paragraph from a given configur ation file.
  _cptrLanguage->GenerateGrammar();                                    // Download given rules to a structure managing the
grammar.
  _cptrLanguage->ReadVocabulary(_szActionFile);                        // Get the vocabulary paragraph from a given configuration
file.
  _cptrLanguage->GetVocabulary(_cptrLanguage->GetGrammar());// With a given grammar, the voc package will be generated.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
             |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being ready for use.
                                  |
| On Error:               FALSE is returned if the instance is not properly configured.
              |
\*-----------------------------------------------------------------------------------------------------------------~*/
char Translator::ReadyToUse(void)
 {
  return _bClassReadyToUse;
 }
/*~-----------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                     |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-----------------------------------------------------------------------------------------------------------------~*/
Translator::~Translator(void)
 {
  #ifdef VERBOSE
```

```
      cout << "Destroying Translator...\n";
 #endif
 if(_cptrLanguage != NULL)                                    // The associations will be dis- ...
  delete _cptrLanguage;                                  // ... connected. This is done with ...
 if(_szActionFile!= NULL)                                     // ... ending of the life-time of ...
       delete [] _szActionFile;                               // ... the owner of that relation: Per def. an association.
}
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| VacModel: Implement the vacuum chamber model.                                                                       |
|                                                                                                                     |
|  File: vacmodel.c                                                                                                   |
|  Running level in design of robot controller software: 2                                                            |
|  Task:                                                                                                              |
|            See model.h; Solving the pure virtual functions of the mode l class. The machine dependend actions are so lved here |
|            with the concrete model, there will be built up a specific action list, representing the concrete model (world) of|
|            the system supported. Theses actions will cause the concrete machine (RobotDevice) to send the hardware dependend |
|            commands, then.                                                                                          |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:                                                                                                            |
|                                                                                                                     |
|                   Date         Reason                                                                               |
|                                                                                                                     |
|                   15.03.1996   Foundation                                                                           |
|                                                                                                                     |
|                   1.0                                                                                               |
|                                                                                                                     |
\*-------------------------------------------------------------------------------------------------------------------~*/


#include "VacModel.h"
#include <iostream.h>
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.   This class is the concrete implement ation of the vacuum chamber model. It     |
|            solves the pure virtual functionality of it’s interface class "ControllerInterface" and is a concrete i mplemen-   |
|            tation of it’s parent Model. Within the model, the action list for current operations is set up. This task list   |
|            is needed within the Run() functionality of the controller object.                                       |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:                                                                                                      |
|                                                                                                                     |
|            none.                                                                                                   |
|                                                                                                                     |
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:                                                                                                          |
|                                                                                                                     |
|            inconspicuous.                                                                                           |
|                                                                                                                     |
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:                                                                                                        |
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
VacModel::VacModel(void *cptrAnchor) : Model(cptrAnchor)
 {
  _bClassReadyToUse=FALSE;                                              // With construction, the members are not initiated
properly.
  #ifdef VERBOSE                                                        // Available in demonstration/debug situations, only. ...
       cout << "Initializing VacModel (concrete)\n";      // ... There will be some output in the standard ...
  #endif                                                                       // ... command window while destruction is perfo r-
med.
  _stActionList.iActions=0;                                             // There are no actions defined, yet. ...
  _stActionList.stDeviceAction = NULL;                    // ... and therefore the list is empty.
  _bClassReadyToUse=TRUE;                                             // All members are initiated; the object is ready for use.
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
              |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, inven ted by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
```

```
| Return value:           TRUE in case of being ready for use.
                                |
| On Error:               FALSE is returned if the instance is not properly configured.
              |
\*-------------------------------------------------------------------------------------------------------------------------------~*/
char VacModel::ClassReadyToUse(void)
 {
  return _bClassReadyToUse;                                                  // Flag indicating the status of object usability.
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Each device of the virtual robot is active or i nactive at specific time. In case, that the time |
|            interval is reached within which the device is to be changed from inact ive to active status, it is taken into the |
|            action list for the current time tick. The decision whether a part of the vi rtual robot (may being a conglomerate |
|            of mentioned devices) is active or not is done within the GetCurrentTa skList functionality. That function uses    |
|            AddDeviceAction to set up an action list for the current interval.
                             |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                  |
| On Error:                       FALSE in case of failure.
                                        |
\*-------------------------------------------------------------------------------------------------------------------------------~*/
char VacModel::AddDeviceAction(int iA)
 {
  DeviceAction *stBuffer;                                                    // Dynamically enlarging the action list by one new
element. ...
  stBuffer= new DeviceAction [_stActionList.iActions+1];  // ... Therefore the new list should be one element larger ...
```

```
  if(stBuffer==NULL) return FALSE;                                        // ... than the old one.
  if(_stActionList.stDeviceAction != NULL)                              // If there are any actions defined already, ...
   for(int iB=0;iB<_stActionList.iActions;iB++)                         // get them into the new list, first.
       stBuffer[iB]=_stActionList.stDeviceAction[iB];                   // ... .
  stBuffer[_stActionList.iActions].iDevice=                             // The new device event is to be ...
                                _stTaskList.stDeviceEvent[iA].iDevice; // ... documented as new ...
  stBuffer[_stActionList.iActions].bActive=TRUE;                 // ... element and marked to be an active (executing) device.
  if(_stActionList.stDeviceAction != NULL)                              // If there is an old list existing, ...
    delete [] _stActionList.stDeviceAction;                             // ... replace it ...
  _stActionList.stDeviceAction=stBuffer;                                // ... with the current one.
  _stActionList.iActions++;                                                     // Finalize new list; make new element acce s-
sible.
  return TRUE;                                                                  // The list is updated, successfu lly.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. With this function, an action list is generated. Cyclic and non cyclic events are
|
|            supported.
                                                      |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           Void pointer casting to the action list.
                                 |
\*-------------------------------------------------------------------------------------------------------------------~*/
void * VacModel::GetCurrentTaskList(long int lCurrentUnit)
 {
  if(_slCycleInterruptor==NULL)                                                 // Defined in Model base class. This member
is needed for...
```

```
   {                                                                                    // ... cyclic actions. With the-
se, a device is switched  ...
    _slCycleInterruptor= new long int [_stTaskList.iEvents+1];   // ... on after start time. The device is activated for  ...
      if(_slCycleInterruptor == NULL)                                                   // ... a defined time span. This is redun-
dant. To keep   ...
       return NULL;                                                                     // ... track on this cyclic
events, an interruptor should...
      for (int iA=0;iA < _stTaskList.iEvents; iA++)                          // ... allways build the offset to the current tick.
The ...
       _slCycleInterruptor[iA]=-1L;                                                     // ... next interruption is defined with
"just finished  ...
   }                                                                                    // ... tick" + next start time +
time span for activation.
  _stActionList.iActions=0;                                                             // No current actions defined.
  if(_stActionList.stDeviceAction != NULL)                                  // An old list should be deallocated.
   {
      delete [] _stActionList.stDeviceAction;
    _stActionList.stDeviceAction = NULL;
   }
  for(int iA=0;iA< _stTaskList.iEvents;iA++)                              // "Events" often are equal to "devices doing an action".
   {                                                                                    // All "Events" which are not
cyclic do have one start time, ...
    if(!_stTaskList.stDeviceEvent[iA].lCyclingPeriod)           // ... and either an stop time or do just start and do not ...
     {                                                                                  // ... have an explicit stop time.

      if(lCurrentUnit >= _stTaskList.stDeviceEvent[iA].lStartActionTick)       // With having a start action tick, ...
         {                                                                                                  // ... the
device is active. As far as...
       if((lCurrentUnit < _stTaskList.stDeviceEvent[iA].lStopActionTick) || // ... it’s end action tick is not reached ...
             (_stTaskList.stDeviceEvent[iA].lStopActionTick == -1))         // ... and in case of not having an end action tick,
...
             if(!AddDeviceAction(iA))                                                   // ... the "Event" is taken into the
action list for ...
              return NULL;                                                                      // ... the current time
tick.
       }
     }
    else                                                                                // If there is a cyclic "Event", the
cycle interruptor for this ...
     {                                                                                  // ... explicit "Event[iA]" is to be initia-
ted. The very first  ...
        if(_slCycleInterruptor[iA] < 0)                                 // ... situation is simply the time interval starting from
the ...
         {                                                                              // ... current tick. After that time interval
is over, the cycle ...
       _slCycleInterruptor[iA]=                                         // ... for this "Event[iA]" is over and will restart at next ...
```

```
                        _stTaskList.stDeviceEvent[iA].lCyclingPeriod;//...start-action-tick, later. From then on, the
"Event[iA]" ...
       }                                                                                    // ... will be active until the
interruptor is reached, again.

      if(lCurrentUnit > _slCycleInterruptor[iA])               // The interruptor for the current cyclic "Event[iA]" is ...
       if(!AddDeviceAction(iA)) return NULL;                   // ... reached and therefore the "Event" will be added to the
list.



       if(lCurrentUnit >=                                                  // With reaching the end of the active inter-
val, ...
            (                                                                                // ... which is defined by
the span within the ...
               _slCycleInterruptor[iA]+                                          // ... interruption and the next stop
tick,
               _stTaskList.stDeviceEvent[iA].lStopActionTick)         // ... the new interruption date is calculated. ...
              )                                                                              // ...
        _slCycleInterruptor[iA]=                                          // ...
                        lCurrentUnit +                                                   // ... Beginning with the cur-
rent tick, ...
                        _stTaskList.stDeviceEvent[iA].lCyclingPeriod +    // ... taking until the active period is over ...
                        _stTaskList.stDeviceEvent[iA].lStopActionTick     // ... and reaching the stop action tick.
                        - 2L;                                             // This correction is caused by the difference of
start counting ...
     }                                                                    // ... with zero instead of "1" (C-Language spec i-
fic) for both periods.
   }

  return (void *) &_stActionList;                                        // Action list is set up for controller use.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. The model gets it’s complete task package from the controller. This functionality pe rforms a    |
|            synchronization of the model with the controller, which did it’s translation, already.
              |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
```

```
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           Void pointer casting to the action list.
                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void VacModel::Synchronize(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Synchronize VacModel with ";             // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while destruction is perfo r-
med.

  _stTaskList=_cptrAnchor->GetTaskPackage();       // Get the complete task pack age from controller.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. It is responsible for solving the pure virt ual::ControllerInterface functionality. This, |
|            to be consistent with that pure abstract interface class. With current design, the Model and View class (Model is |
|            parent of VacModel) share the same interface class, which eases tran sfer operability with the Controller class.
|
|            To implement it’s functionality, a task list of the current situation is ordered, first. With that task list,   |
|            the current task needed to be executed is solved within GetCurrentTaskList, later in the context.
        |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
```

```
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void VacModel::Invalidate(void)
 {
  Synchronize();                                                          // All operations necessary, if this model is out of
focus.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. It is responsible for solving the pure virt ual::ControllerInterface functionality. This, |
|            to be consistent with that pure abstract interface class. With current design, the Model a nd View class (Model is |
|            parent of VacModel) share the same interface class, which eases tran sfer operability with the Controller class.
|
|            To implement it’s functionality, an task list of the current situation is ordered, first. With that task list,
|
|            the current task needed to be executed is solved within GetCurrentTaskList, later in the context. This ve rsion of |
|            Invalidate allows passing a long integer variable for internal use (if needed).     Passing a long parameter is not
|
|            yet needed in the current design, but may become of interest, later.
                    |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
```

```
| Return value:           none.
                                                      |
\*-----------------------------------------------------------------------------------------------------------------------~*/
void VacModel::Invalidate(long int lUnit)
 {
  Synchronize();                                                          // All operations necessary, if this model is out of
focus.
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                    |
|-----------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-----------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-----------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-----------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-----------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
VacModel::~VacModel(void)
 {
  #ifdef VERBOSE                                                          // Available in demonstration/debug situations, only. ...
       cout << "Terminating VacModel.\n";                         // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while construction is perfo r-
med.
  if(_stActionList.stDeviceAction != NULL)               // If there is an action list, ...
   {                                                                      // ... deallocate it.
      delete [] _stActionList.stDeviceAction;
    _stActionList.stDeviceAction = NULL;
   }
 }
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| VacuumChamber: Managing the vacuum chamber control.                                                                 |
|                                                                                                                     |
|  File: vacuumchamber.c                                                                                              |
|  Running level in design of robot controller software: 1                                                            |
|  Task:                                                                                                              |
|            See controller.c; Solving the pure virtual functions of the controller class. The machine "Vacuum Chamber" needs  |
|            to implement hardware dependend extentions like valves, pump and port. These lo gical units are instantiated here. |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:                                                                                                            |
|                                                                                                                     |
|                   Date         Reason                                                                               |
|                                                                                                                     |
|                   15.03.1996   Foundation                                                                           |
|                                                                                                                     |
|                   1.0                                                                                               |
|                                                                                                                     |
\*-------------------------------------------------------------------------------------------------------------------~*/
#include <iostream.h>
#include <ctype.h>
#include <stdlib.h>
#include "VacuumChamber.h"
#include "VacModel.h"
#include "Robot.h"
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.  This class is the concrete implement ation of the controller. It solves the    |
|            pure virtual functionality of it’s abstract base class "Controller" and is a concrete implementation ther efore.   |
|            Within the model, the action list for current operations is set up. This task list is now needed within the Run() |
|            functionality of that controller object.
                                        |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:                                                                                                    |
|                                                                                                                   |
|            none.                                                                                                 |
|                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:                                                                                                        |
|                                                                                                                   |
|            inconspicuous.                                                                                         |
|                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:                                                                                                      |
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------------~*/
VacuumChamber::VacuumChamber(int iArgs,char**szArgs):Controller(iArgs,szArgs)
 {
  _bClassReadyToUse=FALSE;                                                    // With construction, the members are not initiated
properly.
  _stTaskSchedule.stDeviceEvent=NULL;                              // Struct containing a list of device events (defined in Contro l-
ler)
  _stTaskSchedule.iEvents=0;                                                 // Currently, there are not events known.
  _cptrVacModel=NULL;                                                        // There is no relation with a model establi shed.
  _cptrSwatch=NULL;                                                          // There is no relation with an output established.
  _cptrRobot=NULL;                                                           // There is no relation with a virtual robot machine
established.
  _cptrVcDevice=NULL;                                                        // There is no guilty visitor (used with the virtual
robot) existing.
  AssignDevices();                                                           // Each device is announced to the controller with a
unique ID-code.
  _bClassReadyToUse=TRUE;                                                 // ... After finalizing that step, the class is "ready to
use".
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
             |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
```

```
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being ready for use.
                                |
| On Error:               FALSE is returned if the instance is not properly configured.
                |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::ClassReadyToUse(void)
 {
  return _bClassReadyToUse;                                                  // Flag indicating the status of object usa-
bility.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. The devices do get a unique ID-code. This announces them to the controller and allows their     |
|            class global identification.
                                              |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            With the current version, the mnemonics are not flexible. This is a good point for expanding the configur ation    |
|            file with a "PSEUDOCODE" section. That could be read in here and being associated with a dynamically d esigned     |
|            array of devices. That allows to be flexible in supporting machines, which do use a different set of dev ices      |
|            during life-time. With the current version, the pseudocodes  are stored within the header "vacuumchamber.h".
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                              |
\*-------------------------------------------------------------------------------------------------------------------------~*/
```

```
void VacuumChamber::AssignDevices(void)
 {
  _ubValve1=VALVE_1_PSEUDOCODE;
  _ubValve2=VALVE_2_PSEUDOCODE;
  _ubValve3=VALVE_3_PSEUDOCODE;
  _ubPump=PUMP_PSEUDOCODE;
  _ubOmegaRobot=TERMINATE_OMEGA;
  _ubVentilator=VENTILATOR_PSEUDOCODE;
  _ubArretationA=ARRETATION_1_PSEUDOCODE;
  _ubArretationB=ARRETATION_2_PSEUDOCODE;
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Testing the proper initialization of the grammar section within the language package.
        |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                 |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                    |
| On Error:               FALSE in case of failure.
                                |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::TestGrammar(void)
 {
  LanguagePackage stLanguage=GetLanguagePackage();                        // Return struct containing compiled Gra mmar ...
  if(stLanguage.stGrammar._bSeparator == 0) return FALSE;              // ... and vocabulary. Defned in the base class ...
  if(stLanguage.stGrammar._bTimeSymbol == 0) return FALSE;                // ... Controller.
  if(stLanguage.stGrammar._bCycleSymbol == 0) return FALSE;
  if(stLanguage.stGrammar._bOpenParenthesis == 0) return FALSE;
```

```
  if(stLanguage.stGrammar._bCloseParenthesis == 0) return FALSE;
  if(stLanguage.stGrammar._bEnumerator == 0) return FALSE;
  if(stLanguage.stGrammar._szCommentSymbol[0] == '\0') return FALSE;
  if(stLanguage.stGrammar._bDeviceSymbol == NULL) return FALSE;
  if(stLanguage.stGrammar._iDevices == 0) return FALSE;
  return TRUE;
 }
/*~------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Testing the proper initialization of the vcabulary section within the language package.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                     |
| On Error:               FALSE in case of failure.
                                    |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::TestVocabulary(void)
 {
  LanguagePackage stLanguage=GetLanguagePackage();                          // Return struct containing compiled voc abulary ...
  if(stLanguage.stVocabulary._iNumVocListElements==0) return FALSE; //... and gra mmar. Defned in the base class ...
  if(stLanguage.stVocabulary._szVocList == NULL) return FALSE;         // ... Controller.

  return TRUE;
 }


/*~------------------------------------------------------------------------------------------------------------------------*\
```

```
| Task:      Private member. Specify the device ordered with the given language and current vocabulary package.
        |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            With the current version, the idenification values should be added manually, here ("V", "P", "O"). To get more    |
|            flexibility, an algorithm could be implemented, to get these spec ifiers from the configuration file. With that    |
|            a strategy for assotiating mnemonics (like "V" for valve) with the concrete device (like valve 1: "V1") could be  |
|            implemented more flexible within the configuration file. For implement ation, the single member variables (e.g.    |
|            _ubValveX) should be replaced with dynamically designed vectors (couple of: *_ubValve; and it's identifier: iA in |
|            _ubValve[iA]).
                                                          |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                    |
| On Error:               FALSE in case of failure.
                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::GetDevice(int iElement)
 {
  char *szLine=_stLanguage.stVocabulary._szVocList[iElement];    // szLine will contain a complete command, like V3tc(5,1).
  if(szLine[0] == 'V')                                                                                    // Concerning a valve-
device, which could be 1, 2 ...
   {                                                                                                             // ... or three, the
association to the abstract ...
      if(szLine[1] == '1')                                                                                 // ... Device Event is
performed here. For more ...
      _stTaskSchedule.stDeviceEvent[iElement].iDevice=_ubValve1;       // ... flexibility in this point see explanat ions ...
      if(szLine[1] == '2')                                                                                 // ... in section Improve-
ment within this header.
      _stTaskSchedule.stDeviceEvent[iElement].iDevice=_ubValve2;
      if(szLine[1] == '3')
```

```
      _stTaskSchedule.stDeviceEvent[iElement].iDevice=_ubValve3;
   }
  if(szLine[0] == 'P')                                                                    // Concerning a valve-
device.
   _stTaskSchedule.stDeviceEvent[iElement].iDevice=_ubPump;
  if(szLine[0]== 'O')                                                                     // Concerning the whole
machine "Omega".
   _stTaskSchedule.stDeviceEvent[iElement].iDevice=_ubOmegaRobot;
  return ((_stTaskSchedule.stDeviceEvent[iElement].iDevice) ? TRUE : FALSE);
 }

/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member. Install a cycling event to the task schedule.
                           |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                    |
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                    |
| On Error:               FALSE in case of failure.
                                 |
\*-------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::GetCyclingSchedule(int iElement,int iPos)
 {
  char *szLine=_stLanguage.stVocabulary._szVocList[iElement];    // szLine will contain a complete command, like V3tc(5,1).
  if(szLine[iPos++] != _stLanguage.stGrammar._bOpenParenthesis)              // If the grammar needed for that operation, ...
      return FALSE;                                                                                  // ... is undefined,
it is not possible to  ...
```

```
                                                                                                                  // ... understand
the commands.
  if(!isdigit(szLine[iPos]))                                                                          // Keep proof on proper syntax.
       return FALSE;
  _stTaskSchedule.stDeviceEvent[iElement].lCyclingPeriod=                        // Do a numerical excerption of the ...
             GetNumerical(szLine,iPos);                                                                  // ... period information
within the order.
  if(szLine[++iPos] != _stLanguage.stGrammar._bEnumerator)                             // Keep proof on proper informations. ...
       return FALSE;                                                                                        // ...
  if(!isdigit(szLine[++iPos]))                                                                        // ...
       return FALSE;                                                                                        // ... .
  _stTaskSchedule.stDeviceEvent[iElement].lStopActionTick=                             // Do a numerical excerption of the ...
       GetNumerical(szLine,iPos);                                                                     // ... action tracking
informations ...
  _stTaskSchedule.stDeviceEvent[iElement].lStartActionTick=0L;                 // ... for a cycling event.
  #ifdef VERBOSE
  if(_stTaskSchedule.stDeviceEvent[iElement].lCyclingPeriod)                   // Will be a valid signal during Run().
   {
    cout << "\n################################ Cycling event detected ##########################\n";
       cout << "\tCommand:" << szLine << "\n";
    cout << "\tCycling Stop=" << _stTaskSchedule.stDeviceEvent[iElement].lStopActionTick << "\n";
    cout << "\tCycling Start=" << _stTaskSchedule.stDeviceEvent[iElement].lStartActionTick << "\n";
    cout << "\tCycling Cycl=" << _stTaskSchedule.stDeviceEvent[iElement].lCyclingPeriod << "\n\n";
   }
  #endif
  return TRUE;
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member. Install a linear event to the task schedule.
                           |
|--------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|--------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|--------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                    |
|--------------------------------------------------------------------------------------------------------------------|
```

```
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                    |
| On Error:               FALSE in case of failure.
                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::GetTimingSchedule(int iElement,int iPos)
 {
  char *szLine=_stLanguage.stVocabulary._szVocList[iElement];    // szLine will contain a complete command, like V3tc(5,1)....
  if(!isdigit(szLine[iPos])) return FALSE;                                          // Keep proof on proper syntax.
  _stTaskSchedule.stDeviceEvent[iElement].lStartActionTick=                    // Do a numerical excerption of the ...
                                                GetNumerical(szLine,iPos);                        // ... time information
within the order.
  while(isdigit(szLine[++iPos]));                                                          // Be sure to start with the
next valid ...
                                                                                                                  // ... flag,
or if there isn't any, break ...
                                                                                                                  // ... with
the terminating '\0'.
  if(szLine[iPos] != _stLanguage.stGrammar._bEnumerator)                    // Just one argument like: Ot10 which means ...
   {                                                                                                                // ...there is a
start, no end and no cycling.
    _stTaskSchedule.stDeviceEvent[iElement].lStopActionTick=-1L;       // Initiate the corresponding signature ...
    _stTaskSchedule.stDeviceEvent[iElement].lCyclingPeriod=0L;              // ... which has no stop and no cycling ...
      #ifdef VERBOSE                                                                                              // ... information.
                cout << "\n################################ Punctual event detected ########################\n";
                  cout << "\tCommand type A:" << szLine << "\n";
                  cout << "\tA:Event at=" << _stTaskSchedule.stDeviceEvent[iElement].lStartActionTick << "\n\n";
      #endif
   return TRUE;}
 if(!isdigit(szLine[++iPos])) return FALSE;// Keep proof on a proper command signature for other argument types like Vt10,30
  _stTaskSchedule.stDeviceEvent[iElement].lStopActionTick=                  // Get the signature for events with a de-...
                                                GetNumerical(szLine,iPos);                        // ... fined start, stop
and ...
  _stTaskSchedule.stDeviceEvent[iElement].lCyclingPeriod=0L;                // ... no cycle.
  #ifdef VERBOSE
                cout << "\n################################ Linear event detected ########################\n";
                  cout << "\tCommand type B:" << szLine << "\n";
                  cout << "\tB:Linear event stop=" << _stTaskSchedule.stDeviceEvent[iElement].lStopActionTick << "\n";
                  cout << "\tB:Linear event start=" << _stTaskSchedule.stDeviceEvent[iElement].lStartActionTick << "\n\n";
  #endif
  return TRUE;
 }
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member.    Distinguish: linear or cycling events using the given language.
|              |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            With the current version, the indicators for the types of operations are fixed (’t’, ’c’). This is a r eduction for|
|            a simple demonstration, to keep sourcecode description easy. To  implement a more flexible behaviour, the mnemonics|
|            for indication should be fed from the configuration file. An additi onal algorithm could provide a further functio-|
|            nality for a more common and "virtual machine" understanding of these mnemonics. A good place for that point is   |
|            e.g. the vocabulary section.
                                                |
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                      |
| On Error:               FALSE in case of failure.
                                 |
\*-------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::GetSchedule(int iElement)
 {
  int iA=0;
  char *szLine=_stLanguage.stVocabulary._szVocList[iElement];               // szLine will contain a complete command, like ...
                                                                                                                  // ...

V3tc(5,1).
  do {                                                                                                          // Fix type of sche-
dule: Linear time event, ...
   if(szLine[iA++] == ’t’)                                                                                  // ... cyclic dependency,
temperature dependent, etc.
    {                                                                                                           // In case of having
a linear time dependency ...
       _stTaskSchedule.stDeviceEvent[iElement].bEventType=                          // ... the corresponding signature should be
...
```

```
                    DEVICE_EVENT_TYPE_TIMEDEPENDEND;                                    // ... worked out. Starting with bE-
ventType, the ...
        break;                                                                                 // ... completion of that signa-
ture is to be fini-...
    }                                                                                   // ... shed within the detailing sub-
functions like ...
  }while(szLine[iA] != '\0');                                                 // ... GetTimingSchedule() or GetCyclingSchedule(),
...
                                                                                                                    // ... which
specialize the information for the ...
                                                                                                                    // ... opera-
tions of the virtual robot for time ...
                                                                                                                    // ... orien-
ted events.

  if(_stTaskSchedule.stDeviceEvent[iElement].bEventType                          // Time depending events for e.g. valve-
devices.
                    == DEVICE_EVENT_TYPE_TIMEDEPENDEND)
   {
    if(szLine[iA]=='c')                                                                          // Events of cyclic na-
ture.
       GetCyclingSchedule(iElement,iA+1);
      else
       GetTimingSchedule(iElement,iA);                                              // Events which happen for a single
duration or ...
   }                                                                                                  // ... do initiate
once within the whole virtual ...
                                                                                                                    // ... robots
lifetime.

  return ((_stTaskSchedule.stDeviceEvent[iElement].bEventType)               // Return the proper initialization of the ...
                    ? TRUE : FALSE);                                                             // ... corresponding si-
gnature as control flag.
 }


/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Build task schedule. (Solved pure virtual::Controller)
                    |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
```

```
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                        |
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                    |
| On Error:               FALSE in case of failure.
                                  |
\*-----------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::GenerateApplicationParameters(void)
 {
  _stTaskSchedule.stDeviceEvent=                                                    // List of device events. Each of these
events ...
            new DeviceEvent[_stLanguage.stVocabulary._iNumVocListElements];  // ... do contain informations concerning start...
  if(_stTaskSchedule.stDeviceEvent==NULL) return FALSE;                       // ... stop, cycle period, event type, device, etc.
  _stTaskSchedule.iEvents=_stLanguage.stVocabulary._iNumVocListElements;      // There are as much events as there are vocs.
  for(int iA=0;iA < _stTaskSchedule.iEvents; iA++)                            // Initialization of all event signatures ...
   {                                                                                                          // ...
      _stTaskSchedule.stDeviceEvent[iA].bEventType=FALSE;                           // ...
      _stTaskSchedule.stDeviceEvent[iA].iDevice=FALSE;                              // ...
      if(!GetDevice(iA)) return FALSE;                                                 // ... and finally definition of
the events, ...
    if(!GetSchedule(iA)) return FALSE;                                                 // ... used with the virtual ro-
bot.
   }
  return TRUE;
 }
/*~-----------------------------------------------------------------------------------------------------------------*\
| Task:     Protected member. Get control on semantic and syntax of the task schedule. (Solved pure virt ual::Controller).  |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
```

```
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            With the current version, no algorithm is installed for that purpose. This should be implemented when all other   |
|            design-decisions concerning implementation of more flexibility with dev ices are done. This point will be directly |
|            dependend on those decisions. With the current versio n, there is no need for an explicit testing of these appli-  |
|            cation parameters, because the complexity did not reach that level which would force to do so.
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
| On Error:               FALSE in case of failure.
\*-------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::TestApplicationParameters(void)
 {
  return TRUE;                                                                                    // See secti-
on "improvement" within this header.
 }


/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Instantiate vacuum chamber model. (Solved pure virtual::Controller).
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            No suggestions.
```

```
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                        |
| On Error:               FALSE in case of failure.
                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::LaunchModel(void)
 {
  _cptrVacModel= new VacModel((void *) this);                          // Get an anchor to the controller object during runtime.
...
  if(_cptrVacModel == NULL) return FALSE;                              // ... casting to void * eases using the controller inter-
...
  return TRUE;                                                                                  // ... face for all possible models.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Instantiate vacuum chamber views. (Solved pure virtual::Controller).
             |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            With the current version, there is no implementation for performing a dynamic initialization of the DevP arams     |
|            structure. Using the information t o be found in the configuration file could increase the flexibility wi thin this |
|            funcionality.
                                                          |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                       |
```

```
| On Error:               FALSE in case of failure.
                                  |
\*-------------------------------------------------------------------------------------------------------------------~*/
char VacuumChamber::LaunchViews(void)
 {
  DevParams stDevParms;                                                    // Declaring port, device type, communication str a-
tegy, etc.
  stDevParms.iPort=ROB_LPT1;                                               // Using the LPT1 as port for the vacuum chamber
interface. ...
                                                                                  // ... Other possible ports are:
LPTn and COMn (which are ...
                                                                                  // ... DEV_PORTS, too).
  stDevParms.bTypeOfDevice=DEV_PORT;                                  // Using a direct communication strategy within the VcD e-
vice.
                                                                                  // ... Other possible strategies
are: DDE and Ne twork.
  _cptrVcDevice= new VcDevice(stDevParms);                            // The device is a guilty visitor, used with Robot.
  if(_cptrVcDevice==NULL) return FALSE;                                    // Keep track on some possible "weak-memory" situ a-
tions.
  _cptrRobot=new Robot((void *) this, _cptrVcDevice);           // The robot object (as a 100% view) is using a relation ...
  if(_cptrRobot==NULL) return FALSE;                                  // ... to solve the hardware specific operations. The o b-
ject...
  return TRUE;                                                                   // ... supporting that hardware depe n-
dency is located within ...
                                                                                  // ... the VcDevice object,
being instanciated above. Keep ...
                                                                                  // ... attention to the fact,
that VcDevice is not a view of ...
                                                                                  // ... Controller. It is an a u-
xiliary object, used within a ...
                                                                                  // ... relation relative to the
Robot object.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Cause views to recall. (Solved pure virt ual::Controller).This member function causes a synchro- |
|            nization of the virtual robot’s current situation with the current timing. An invalidation causes the vi rtual     |
|            robot system to check it’s current action database and  react on the needs documented there. This is perfo rmed with|
|            the interface member "Invalidate" of the Robot object.
                                  |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
```

```
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                        |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of success.
                                        |
| On Error:               FALSE in case of failure.
                                  |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void VacuumChamber::InvalidateViews(void)
 {
  _cptrRobot->Invalidate();                                                    // Cause robot to synchronize to the current
timing.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. Cause views to recall. (Solved pure virt ual::Controller).This member function causes a synchro- |
|            nization of the virtual robot's current situation with the current timing.  An invalidation causes the virtual     |
|            robot system to check it's current action database and react on the needs documented there. This is perfo rmed with|
|            the interface member "Invalidate" of the Robot object.
                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                        |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
```

```
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                          |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void VacuumChamber::InvalidateViews(long int lUnit)
 {
  _cptrRobot->Invalidate(lUnit);                                          // Passing the current tick in form of a long local me m-
ber.
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Protected member. The model needs to get it's complete task package from this controller. Calling the inv alidation|
|            function of VacModel's interfac e performs a synchronization of that model with this controller, which o ffers the  |
|            properly translated information, already.
                                      |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                          |
\*-------------------------------------------------------------------------------------------------------------------------~*/
void VacuumChamber::ConfigureModel(void)
 {
  if(_cptrVacModel != NULL)                                                    // If instantiation of the vacuum chamber's
model succeeded,...
   _cptrVacModel->Invalidate();                                              // ... cause it's synchronization with the inform a-
tion ...
```

```
                                                                                // ... offered with this con-
troller object.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Make the vacuum chamber active. (Solved pure virtual::Controller ). This functionality is |
|            called from alien classes to get the virtual robot up and running. After an alien object did an instanti ation of  |
|            this class, it will call the Run functionality, soon. With an already existing Language and translated task
|
|            schedule, all models and views will be launched. Causing a synchronizat ion of all observers will finally lead to  |
|            the stepwise working off of all tasks to be done by the virtual robot. The signals which cause to step the "heart"|
|            forward may be different. In case of the vacuum chamber controller it is a timer, causing events each s econd.     |
|            Other robots (temperature controller) may be informed with reaching or leaving temperature ranges.
        |
|            With this version of demonstration, a simple keyboard event will cause the stepping. This is implemented to easily|
|            follow messages on screen.
                                                  |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                        |
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                               |
\*-------------------------------------------------------------------------------------------------------------------~*/
void VacuumChamber::Run(void)
 {
  if(!TestGrammar()) return;
  if(!TestVocabulary()) return;
  if(!GenerateApplicationParameters()) return;
  if(!TestApplicationParameters()) return;
  if(!LaunchModel()) return;
```

```
  if(!LaunchViews()) return;
  ConfigureModel();
  long int lTick=0L;
  for(int iA=0; iA < 10; iA++)
   cout << "\n";
  cout << "\t/------------------------------------------------------------\\\n";
  cout << "\t| Each following return will simulate an event which will  |\n";
  cout << "\t|                              have                        |\n";
  cout << "\t|  the virtual robot to proceed with it's time schedule.   |\n";
  cout << "\t|__________________________________________________________|\n";
  cout << "\t|Press return to continue or <Ctrl-C> to break immediately.|\n";
  cout << "\t\\------------------------------------------------------------/\n";
  for(iA=0; iA < 10; iA++)
   cout << "\n";
  for(;;)                                        // Simulates timer events
   {
    _stActionList= (ActionList *) _cptrVacModel->GetCurrentTaskList(lTick);
      if(_stActionList == NULL)
       {
        cout << "Robot terminates." << "\n";
        return;
       }

    InvalidateViews(lTick);            // The views will then synchronize with the information they need at lTick unit.
      if(!_cptrVcDevice->RobotDeviceRunning())
       {
        cout << "Virtual robot terminates." << "\n";
        break;
       }
      cout << "(Press return to continue)";
      cout << "or <Ctrl-C> to break immediately.\n";
    cin.get();
    lTick++;
   }
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task: If there are allocated resources, the destructor will free them.
|
|--------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|--------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
```

```
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                    |
\*-----------------------------------------------------------------------------------------------------------------~*/
VacuumChamber::~VacuumChamber(void)
 {
  if(_cptrVacModel != NULL)                                                    // Terminate all relations. ...
   delete _cptrVacModel;                                                 // ...
                                                                                    // ...
  if(_cptrRobot != NULL)                                                // ...
   delete _cptrRobot;                                                           // ...
                                                                                    // ...
  if(_cptrVcDevice!=NULL)                                               // ...
   delete _cptrVcDevice;                                                // ... .


  #ifdef VERBOSE
  cout << "Leaving VacuumChamber (Controller sibling)\n";
  #endif
 }
/*~-----------------------------------------------------------------------------------------------------------------*\
| VcDevice: Implement the controller event dependend actions for a vacuum chamber device.
|
|
|
|  File: vcdevice.h
|
|  Running level in design of robot controller software: 1                                                          |
|  Task:                                                                                                            |
|            See robotdevice.h; This class is responsible for solving all hardware d ependend operations of the concrete machine|
|            of type "Vacuum Chamber", which consists of 3 v alves, one pump, one doc king station (tip rack at biomek 2000),    |
|            one ventilator (part of the drying station) and one heating element (part of the drying station). The d irect      |
|            access is processed with the electronic interface developed and described within the dissertation.
        |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
```

```
| Explanations for Layout and types see File: controller.h                                                                |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#include "VcDevice.h"
#include "iostream.h"
#include "io.h"
#ifdef VERBOSE
 #include "VacuumChamber.h"
#endif
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.  Passed parameter reports the given hardware dependency of the device.      |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            With the current version, the vacuum chamber needs to have direct communication to the parallel port. To keep this|
|            version simple (demonstration of the design), the additional possibilities for communication were left out. If    |
|            other "ports" were needed, the place to choose them is documented below. See switch(stParams.iPort) for that pur- |
|            pose.
                                                    |
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-------------------------------------------------------------------------------------------------------------------~*/
VcDevice::VcDevice(DevParams stParams) : RobotDevice(stParams)
```

```
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situat i-
ons, only. ...
       cout << "which is a VcDevice in this case.\n";             // ... There will be some output in the standard ...
  #endif                                                                  // ... command window while construct i-
on is performed.
  _bClassReadyToUse=FALSE;                                          // Private flag, indicating proper initi a-
lization of the instance.
  _iPort = -1;                                                            // Initialization to be sure getting a
proper error code ...
                                                                          // ... with the "low level" open li b-

func.
  #ifndef SIMULATION                                                // Do not open a port, if the software runs
on simulation.
   char szPort[20];
   switch(stParams.iPort)
    {
     case ROB_LPT1:                                                 // The vacuum chamber needs to have a direct
way of ...
                               strcpy(szPort,"PRN");                // ... communication to the parallel port. If
there ...
                               break;                                     // ... is a need to have other ways of
communication, ...
     default: return;                                               // ... these "ports" can be chosen, here.
    };
  if(stParams.bTypeOfDevice == PORT)                                // Having a direct communication, the corresp o nding ...
   {                                                                      // ... port is to be opened.
    _iPort=_open(szPort,_O_BINARY | _O_WRONLY );
    if(_iPort <= 0) return;
   }
  _bClassReadyToUse=TRUE;
  #endif
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any. Closing an open port, if used.
        |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
```

```
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-------------------------------------------------------------------------------------------------------------------------------~*/
VcDevice::~VcDevice(void)
 {
  if(_iPort != -1)                                                        // The communication strategy becomes obvious with
checking ...
    _close(_iPort);                                                       // ... the corresponding handles and/or signatures.
With this ...
  #ifdef VERBOSE                                                          // ... demonstration software, the direct communic a-
tion to a ...
   cout << "Leaving VcDevice\n";                                  // ... port is shown, only. Other strategies need to have ...
  #endif                                                                        // ... some additional functionality (like
e.g. closing DDE talk).
 }
/*~-------------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Indicate complete instantiation and initialization by alien query.
             |
|-------------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------------|
| History:
|
```

```
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           TRUE in case of being ready for use.
                                  |
| On Error:               FALSE is returned if the instance is not properly configured.
               |
\*-------------------------------------------------------------------------------------------------------------------~*/
char VcDevice::DeviceReadyForOperation(void)
 {
  return _bClassReadyToUse;                                    // Private flag, indicating proper initialization of the insta n-
ce.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Adaptor struc turing pattern. Solved pure virtual::RobotDevice. This specific op eration   |
|            functionality is the "door" to the real device. With execution, the concrete orders will go out to the co ncrete   |
|            hardware (like e.g. the vacuum chamber interface). With this simulat ion software, a descriptive output into a     |
|            standard terminal screen is done to make the corresponding actions visible,
                     |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                                  |
\*-------------------------------------------------------------------------------------------------------------------~*/
void VcDevice::SpecificOperation(ActionList *stList)
 {
  #ifdef VERBOSE
  static unsigned int suiTick=0;cout << "\n/-------------------------------------------------------------------------\\\n";
```

```
  cout <<  "| VcDevice: Specific operation for vacuum chamber.                    |\n";
  cout << "|                                                                    |\n";
  cout << "| Current task\t" << suiTick++ << "\t\t\t\t\t\t\t|\n";
  for(int iA=0;iA < stList->iActions;iA++)
    {
        switch (stList->stDeviceAction[iA].iDevice)
         {
          case PUMP_PSEUDOCODE: cout << "|\t\t\tPump running.\t\t\t\t\t|\n";break;
          case VALVE_1_PSEUDOCODE: cout << "|\t\t\tValve 0 op ened\t\t\t\t\t|\n";break;
          case VALVE_2_PSEUDOCODE: cout << "|\t\t\tValve 1 op ened\t\t\t\t\t|\n";break;
          case VALVE_3_PSEUDOCODE: cout << "|\t\t\tValve 2 opened (Slick vacuum d estruction).\t|\n";break;
          case TERMINATE_OMEGA: cout << "|\t\t\tAll valves closed. Pump off.\t\t\t|\n";
                    cout << "|\t\t\tSet flag for terminating virtual robot.\t\t|\n";
                    _bOmegaSystemRunning=FALSE;
                    break;
         }
    }
  cout << "\\----------------------------End current task.-------------------------/\n";
  cout << "\n\n\n";
  #endif
  #ifndef SIMULATION                                                                  // Having no simulation, the
commands ...
   char bMnemonic;                                                                    // ... will be sent to the acti-
ve port.
   unsigned int uiToWrite=1;                                                        // The corresponding mnemonics are ...
   for(int iA=0;iA < stList->iActions;iA++)                                  // ... documented within the device ...
    {                                                                                 // ... action list. The
port was opened ...
     bMnemonic=(char) stList->stDeviceAction[iA].iDevice;             // ... during object instantiation within ..
       if(_iPort != -1).                                                              // ... the constructor, which is
in this ...
      _write( _iPort, &bMnemonic,uiToWrite)) == -1 )                        // ... simple case a direct communicative ...
    }                                                                                 // ... strategy. Other
forms of communication ...
  #endif                                                                              // ... (DDE, network) do
get another algorithm,...
                                                                                             // ... which would
be placed here.
 }
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| VacModel: Implement the controller event dependend actions.
                            |
|
|
|  File: view.c
|
|  Running level in design of robot controller software: 1                                                            |
|  Task:                                                                                                              |
|            See controllerinterface.h; All controller interface dependend calls and callbacks are located here.  Normally the  |
|            controller will build up an association with the view class to connect a device to it's (time dependend) infor-   |
|            mational system. The view class is a base class to guarantee an inh erited ControllerInterface on one hand and the |
|            proper connection of the explicit device on the other. This solves the implementation of an abstract fact ory- and |
|            strategy pattern.
                                                    |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                   Date         Reason
|
|                   15.03.1996   Foundation
|
|                   1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#include <iostream.h>
#include "View.h"
#include "Declarations.h"
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.  Receive anchor class (Controller) thereby.
              |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
View::View(void *cptrAnchor) : ControllerInterface()
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
        cout << "Initializing View";                               // ... There will be some output in the standard ...
        cout << " (Abstract Baseclass)...\n";             // ... command window while construction is performed.
  #endif
  _cptrAnchor= (Controller *) cptrAnchor;                 // Get a relation to the active controller object.
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                    |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
View::~View(void)
```

```
 {
  #ifdef VERBOSE                                                  // Available in demonstration/debug situations, only. ...
       cout << "Terminating View...\n";                       // ... There will be some output in the standard ...
  #endif                                                             // ... command window while destruction is perfo r-
med.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Vocabulary: Compiling a vocabulary package.
                                    |
|
|
|  File: vocabulary.c
|
|  Running level in design of robot controller software: 1                                                            |
|  Task:                                                                                                              |
|            With a given configuration file, the vocabulary for the different actions has to be interpreted. The result is a  |
|            package of an executable vocabulary, which can be used for translation into the language used with the a bstract   |
|            machine. Result of the instance of this class will be an encapsulated package, which is accessible with a given   |
|            interface function. Together with a compiled grammar package, the langu age system is complete and ready for being |
|            used with the translator instance which will implement a language object (implemented as a mixin class co mposed of|
|            grammar and vocabulary classes).
                                   |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| Explanations for Layout and types see File: controller.h                                                            |
|- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -|
| History:
|
|                  Date         Reason
|
|                  15.03.1996   Foundation
|
|                  1.0
|
\*-------------------------------------------------------------------------------------------------------------------~*/
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include "Declarations.h"
#include "Vocabulary.h"
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task: Initialization of member variables.
                                   |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
```

```
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:    none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
Vocabulary::Vocabulary(void)
 {
  #ifdef VERBOSE                                                    // Available in demonstration/debug situations, only. ...
       cout << "Initializing Vocabulary.\n";           // ... There will be some output in the standard ...

  #endif                                                                  // ... command window while construction is perfo r-
med.
  _stVocabulary._iNumVocListElements=0;                        // Encapsulated package of ...
  _stVocabulary._szVocList=NULL;                             // ...executable vocabulary.
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Task: Leaving instance. Cleaning up all allocations, if there are any.
                   |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
|
\*-----------------------------------------------------------------------------------------------------------------------~*/
Vocabulary::~Vocabulary(void)
 {
  #ifdef VERBOSE                                                        // Available in demonstration/debug situations, only. ...
       cout << "Terminating Vocabulary...\n";                  // ... There will be some output in the standard ...
  #endif                                                                // ... command window while destruction is perfo r-
med.
  if(_stVocabulary._szVocList != NULL)                                  // In case of an existing vocabulary...
   {                                                                                  // ... the 2dim vector is to be
...
    for(int iA=0;iA< _stVocabulary._iNumVocListElements;iA++)    // ... freed.
       delete [] _stVocabulary._szVocList[iA];                          // ...
      delete [] _stVocabulary._szVocList;                                     // ... .
   }
 }
/*~-----------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member. Auxiliary: enlarging a 2 dim vector (part of voc. package) by one element.
        |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
                                                    |
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
```

```
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                              |
\*-------------------------------------------------------------------------------------------------------------------------~*/

void Vocabulary::AddToVocList(char *szCmd)
 {
  char **szNewVocList;
  szNewVocList= new char *[_stVocabulary._iNumVocListElements+1];
  for(int iA=0;iA < _stVocabulary._iNumVocListElements; iA++)
    szNewVocList[iA]=_stVocabulary._szVocList[iA];
  delete [] _stVocabulary._szVocList;
  _stVocabulary._szVocList=szNewVocList;
  _stVocabulary._szVocList[_stVocabulary._iNumVocListElements]=szCmd;
  _stVocabulary._iNumVocListElements++;
 }
/*~-------------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Get the ACTIONS section from a given configur ation file. This is the "vocabulary" of that|
|            robot; means that these are the "sentences" which will be understood with the grammar and "words" being documented|
|            whithin the GRAMMAR section of the configuration file.
                                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           none.
                                              |
\*-------------------------------------------------------------------------------------------------------------------------~*/
```

```
void Vocabulary::ReadVocabulary(char *szFile)
 {
  char *szBuf;
  char *szNewCmd;
  char bVocabulary=FALSE;
  ifstream smCfgFile;
  smCfgFile.open(szFile,ios::in);
  if(!smCfgFile)
   cout << "Datei" << szFile << "ist nicht existent.";
  smCfgFile.seekg(0,ios::end);               // Reset to end of file
  int iFileSize=smCfgFile.tellg();           // Get the file size.
  smCfgFile.seekg(0,ios::beg);               // Reset to the start of the file.
  if(iFileSize > MAX_CFGFILESIZE) return;
  szBuf= new char[iFileSize+1];
  if(szBuf == NULL) return;
  while(smCfgFile)
   {
    smCfgFile.getline(szBuf,iFileSize,'\n');
       if(!smCfgFile) break;                              // Just EOF is not an element of Language.

    if(!strcmp("GRAMMAR:",szBuf))
        bVocabulary=FALSE;
       if(!strcmp("ACTIONS:",szBuf))
        bVocabulary=TRUE;

       if(bVocabulary)
        {
         if(bVocabulary > TRUE)                     // Excerpt the rules containing lines, only.
          {
              szNewCmd= new char [strlen(szBuf)+1];
              strcpy(szNewCmd,szBuf);

           AddToVocList(szNewCmd);
          }
         else
          bVocabulary++;
        }
   }
  cout << "\n/--------------------------------------------------------------------------\\\n";
  cout <<  "| Vocabulary: Describing actions.\t\t\t\t\t|\n";
  cout << "|                                                                          |\n";
   for(int iA=0;iA < _stVocabulary._iNumVocListElements; iA++)
        cout << "|" << _stVocabulary._szVocList[iA] << "\t|\n";
  cout << "\\-----------------------------End actions described.--------------------/\n";
  cout << "\n\n\n";
  delete [] szBuf;
 }
```

```
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Private member. Manager function for element interpretation of an u nknown command. The result of that function is |
|            a string which is pure "robot vocabulary based". All human unde rstandable comments for example are taken out. The |
|            interpretation is done on the currently guilty grammar, which should be known, therefore.
        |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            Checking the integrity of the grammar package before use.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                 |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           Elements of the purified command, using the guilty grammar.
                            |
\*-------------------------------------------------------------------------------------------------------------------~*/
int Vocabulary::GetElements(char *szLine,GrammarPackage stGrammar,char *szPurified)
 {
  int iA=0;                                                                          // Using the currently guilty grammar,
...
  int iElements=0;                                                                 // ... all elements of a command will be t e-
sted.
  do{
       if(
            (szLine[iA] == stGrammar._szCommentSymbol[0]) &&    // The command may be composed of a valid robot la nguage ...
            (szLine[iA+1] == stGrammar._szC ommentSymbol[1])           // ... and a part which is a comment in human langu age.
         ) break;
       if(szLine[iA] != stGrammar._bSeparator)                           // Purifying the ...
        if(szLine[iA] != ' ')                                                  // ... robot specific part ...
         if(szLine[iA] != ASCII_TAB)                                     // ... of the corresponding command.
          {                                                                          // ...
              if(szPurified != NULL)                                               // ...
               szPurified[iElements]=szLine[iA];                           // ...
        iElements++;                                                           // ...
```

```
          }                                                                                    // ...
  }while(szLine[iA++] != '\0');                                                  // ... .
  if(szPurified != NULL)                                                         // The purified string and its length found in that
...
              szPurified[iElements]='\0';                                        // ... investigation is saved and returned to the
...
  return iElements;                                                                    // ... calling routine.
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Alien objects access the encapsulated vocab ulary package. With an already existing       |
|            grammar a purified vocabulary package will be generated. "Purified" means, that within the vocabulary package     |
|            only 100% robot language based (and therefore "robot-understandable" la nguage) information is stored. There are no|
|            comments, or other elements existing, anymore.
                                    |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                  |
|-------------------------------------------------------------------------------------------------------------------|
| Return value:           Elements of the purified command, using the guilty grammar.
                            |
\*-------------------------------------------------------------------------------------------------------------------~*/
VocabularyPackage Vocabulary::GetVocabulary(GrammarPackage stGrammar)
 {
  char *szBuffer;
  #ifdef VERBOSE
cout <<  "| Purified vocabulary.\t\t\t\t\t\t\t|\n";
  #endif
  for(int iA=0;iA < _stVocabulary._iNumVocListElements;iA++)                     // All vocabulatory elements ...
   {                                                                                                    // ... will
be purified ...
```

```
      szBuffer=                                                                                                  // ... one
after the other, ...
            new char                                                                                             // ... using
the currently ...
                  [GetElements(_stVocabulary._szVocList[iA],stGrammar,NULL)+1];    // ... guilty grammar.
      if(szBuffer != NULL)                                                                          // ... To perform
the operation, ...
       {                                                                                                                 // ...
GetElements will be run ...
        GetElements(_stVocabulary._szVocList[iA],stGrammar,szBuffer);              // ... twice. First, the length ...
        delete [] _stVocabulary._szVocList[iA];                                          // ... of the buffer is deter-
mined, ...
        _stVocabulary._szVocList[iA]= szBuffer;                                          // ... (passing NULL as szBuf-
fer), ...
       }                                                                                                                 // ...
and then finally the infor- ...
      #ifdef VERBOSE                                                                                             // ... mation
is stored in a second call.
       cout << "|\t\t" << _stVocabulary._szVocList[iA] << "\t\t\t\t\t\t";
       if (strlen(_stVocabulary._szVocList[iA]) < 8)
        cout << "\t";
       cout << "|\n";
    #endif
   }
  #ifdef VERBOSE
    cout << "\\--------------------------End purified vocabulary.--------------------/\n";
  #endif
  return _stVocabulary;
 }
/*~-------------------------------------------------------------------------------------------------------------------*\
| Task:      Member of the interface. Alien objects access the encapsulated vocabulary package. This is a simple facil ity to   |
|            access the private member variable, storing the purified robot vocabulary.
                    |
|-------------------------------------------------------------------------------------------------------------------|
| Bugs reported:
|
|            none.
|
|-------------------------------------------------------------------------------------------------------------------|
| Behaviour:
|
|            inconspicuous.
|
|-------------------------------------------------------------------------------------------------------------------|
| Improvement:
|
```

```
|            no suggestions.
|
|-------------------------------------------------------------------------------------------------------------------------|
| History:
|
|            Foundation:  15.03.1996   (robot controller software, invented by Gerhard Kauer)
                   |
|-------------------------------------------------------------------------------------------------------------------------|
| Return value:           Elements of the purified command, using the guilty grammar.
                            |
\*-------------------------------------------------------------------------------------------------------------------------~*/
VocabularyPackage Vocabulary::GetVocabulary(void)
 {
  return _stVocabulary;
 }
```

9.1.3 Konfigurationsdatei

Datei: VC.CFG

```
GRAMMAR:
SEP:-
TIM:t
CYC:c
PAR:()
ENU:,
DEV:V
DEV:P
DEV:O
COM://
ACTIONS:
-V1t5,15    // Open valve 1 after 5 seconds for 15 sec’s duration.
-V3tc(5,1)  // Open valve 3 all 5 seconds for 1 second duration.
-Pt0        // Start vacuum pump with second 0 (immediately).
-Ot15       // Terminate "virtual robot" with reaching second 15.
```

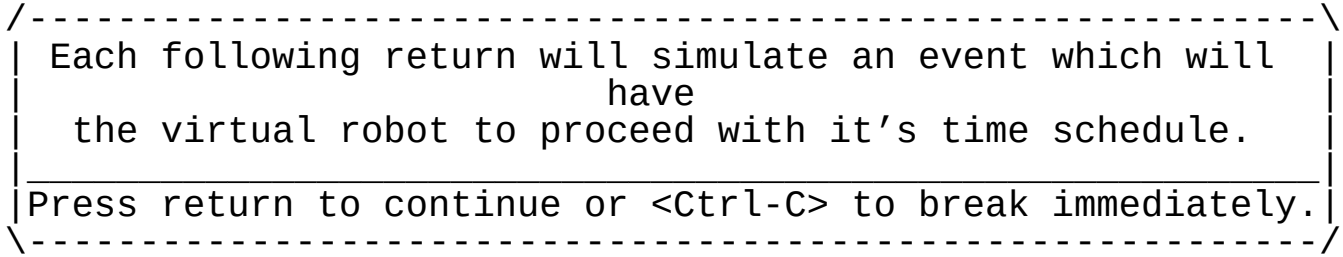
9.1.4 Ausgabe Terminalprogramm

```
################################## Main function ##############################
Running virtual robot with configuration file << vc.cfg
################################## Make new VacuumChamber ####################
Initializing Controller...
Controller: "TranslateArguments..."
Initializing Translator.
Initializing Grammar...
Initializing Vocabulary.
Initializing Language.
Grammar: Reading grammar section of profiling file system.
         Using <vc.cfg>  (334) Bytes
Grammar: Adding 1nd informational package.
Grammar: Adding 2nd informational package.
Grammar: Adding 3nd informational package.
Grammar: Adding 4nd informational package.
Grammar: Adding 5nd informational package.
Grammar: Adding 6nd informational package.
Grammar: Adding 7nd informational package.
Grammar: Adding 8nd informational package.
Grammar: Adding 9nd informational package.
Grammar: 3 Devices found
Grammar: Separator symbol= -
Grammar: t Time Symbol found
Grammar: Cycling symbol= c
Grammar: Parenthesis open= (
Grammar: Parenthesis close= )
Grammar: Enumerator= ,
Grammar: Device symbol(0)= V
Grammar: Device symbol(1)= P
Grammar: Device symbol(2)= O
Grammar: Comment symbol= //
```

```
/-----------------------------------------------------------------------\
| Vocabulary: Describing actions.                                       |
|                                                                       |
|-V1t5,15   // Open valve 1 after 5 seconds for 15 sec’s duration.      |
|-V3tc(5,1) // Open valve 3 all 5 seconds for 1 second duration.        |
|-Pt0       // Start vacuum pump with second 0 (immediately).          |
|-Ot15      // Terminate "virtual robot" with reaching second 15.       |
\----------------------------End actions described.---------------------/


Grammar:Returning grammar package...

/-----------------------------------------------------------------------\
| Purified vocabulary.                                                  |
|           V1t5,15                                                     |
|           V3tc(5,1)                                                   |
|           Pt0                                                         |
|           Ot15                                                        |
\--------------------------End purified vocabulary.---------------------/




Grammar:Returning grammar package...
Language: Returning language
Controller: Specific language transferred to robot’s world.
Destroying Translator...
Leaving Language...
```

```
Terminating Vocabulary...
Grammar: On destruction.
################################ Run VacuumChamber ########################
Controller:"GetLanguagePackage"
Controller:"GetLanguagePackage"
Controller: "GetNumerical"
Controller: "GetNumerical"

################################ Linear event detected ####################
      Command type B:V1t5,15
      B:Linear event stop=15
      B:Linear event start=5

Controller: "GetNumerical"
Controller: "GetNumerical"

################################ Cycling event detected ###################
      Command:V3tc(5,1)
      Cycling Stop=1
      Cycling Start=0
      Cycling Cycl=5

Controller: "GetNumerical"

################################ Punctual event detected ##################
      Command type A:Pt0
      A:Event at=0

Controller: "GetNumerical"

################################ Punctual event detected ##################
      Command type A:Ot15
```

```
      A:Event at=15

Initializing ControllerInterface.(being abstract)
Initializing Model (abstract base class)
Initializing VacModel (concrete)
Initializing RobotDevice...which is a VcDevice in this case.
Initializing ControllerInterface.(being abstract)
Initializing View (Abstract Baseclass)...
Initializing Robot...
Synchronize VacModel with Controller:"GetTaskPackage"
```




```
0: Robot invalidated
```

```
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    0                                                 |
|                 Pump running.                                     |
\---------------------------End current task.----------------------/



(Press return to continue)or <Ctrl-C> to break immediately.
1: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
       with getting the current abstract action and
       solving it with the associated specific operation,
       which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    1                                                 |
|                 Pump running.                                     |
\---------------------------End current task.----------------------/
```

```
(Press return to continue)or <Ctrl-C> to break immediately.
2: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    2                                                 |
|                 Pump running.                                     |
\----------------------------End current task.----------------------/



(Press return to continue)or <Ctrl-C> to break immediately.
3: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
       with getting the current abstract action and
       solving it with the associated specific operation,
       which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    3                                                 |
|                 Pump running.                                     |
\----------------------------End current task.----------------------/
```

```
(Press return to continue)or <Ctrl-C> to break immediately.
4: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    4                                                 |
|                 Pump running.                                     |
\-----------------------------End current task.---------------------/


(Press return to continue)or <Ctrl-C> to break immediately.
5: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    5                                                 |
|                 Valve 0 opened                                    |
```

```
|                  Pump running                                          |
\----------------------------End current task.-----------------------/


(Press return to continue)or <Ctrl-C> to break immediately.
6: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:
/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    6                                                 |
|                 Valve 0 opened                                    |
|                 Valve 2 opened (Slick vacuum destruction).        |
|                 Pump running.                                     |
\----------------------------End current task.-----------------------/


(Press return to continue)or <Ctrl-C> to break immediately.
7: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
```

```
| VcDevice: Specific operation for vacuum chamber.                      |
|                                                                       |
| Current task    7                                                     |
|                 Valve 0 opened                                        |
|                 Pump running.                                         |
\---------------------------End current task.-----------------------/



(Press return to continue)or <Ctrl-C> to break immediately.
8: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-----------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                      |
|                                                                       |
| Current task    8                                                     |
|                 Valve 0 opened                                        |
|                 Pump running.                                         |
\---------------------------End current task.-----------------------/



(Press return to continue)or <Ctrl-C> to break immediately.
9: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
```

```
      solving it with the associated specific operation,
      which is now:
```

```
/-------------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                        |
|                                                                         |
| Current task    9                                                       |
|                 Valve 0 opened                                          |
|                 Pump running.                                           |
\-----------------------------End current task.-------------------------/
```

```
(Press return to continue)or <Ctrl-C> to break immediately.
10: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:
```

```
/-------------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                        |
|                                                                         |
| Current task    10                                                      |
|                 Valve 0 opened                                          |
|                 Pump running.                                           |
\-----------------------------End current task.-------------------------/
```

```
(Press return to continue)or <Ctrl-C> to break immediately.
11: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    11                                                |
|                 Valve 0 opened                                    |
|                 Valve 2 opened (Slick vacuum destruction).        |
|                 Pump running.                                     |
\----------------------------End current task.----------------------/


(Press return to continue)or <Ctrl-C> to break immediately.
12: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:
```

```
/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    12                                                |
|                 Valve 0 opened                                    |
|                 Pump running.                                     |
\-----------------------------End current task.---------------------/


(Press return to continue)or <Ctrl-C> to break immediately.
13: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    13                                                |
|                 Valve 0 opened                                    |
|                 Pump running.                                     |
\-----------------------------End current task.---------------------/


(Press return to continue)or <Ctrl-C> to break immediately.
14: Robot invalidated
```

```
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:

/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
| Current task    14                                                |
|                 Valve 0 opened                                    |
|                 Pump running.                                     |
\---------------------------End current task.----------------------/



(Press return to continue)or <Ctrl-C> to break immediately.
15: Robot invalidated
Controller:"GetCurrentAction"
Robot: synchronizes active devices
      with getting the current abstract action and
      solving it with the associated specific operation,
      which is now:




/-------------------------------------------------------------------\
| VcDevice: Specific operation for vacuum chamber.                  |
|                                                                   |
```

```
| Current task    15                                                  |
|                 Pump running.                                       |
|                 All valves closed. Pump off.                        |
|                 Set flag for terminating virtual robot.             |
\----------------------------End current task.-------------------------/


Virtual robot terminates.
Terminating VacModel.
Leaving Model (abstract base class)
Leaving ControllerInterface(being abstract)
Destroying Robot...
Terminating View...
Leaving ControllerInterface(being abstract)
Leaving VcDevice
Terminating RobotDevice.
Leaving VacuumChamber (Controller sibling)
Leaving Controller...
```

10 Anhang B: Bauteilliste für das Interface zur Ansteuerung der Vakuumkammer

1. Kondensatoren:

	1500µF	
	2200µF	(2)
	100 µF	
	10µF	(3)
	0.1µF	(5)
	0.47µF	
	0.27µF	
	47µF	
	500pF	

2a. Widerstände (fest):

	4.7 kΩ	(3)
	1 kΩ	
	680 Ω	
	470 Ω	
	150 Ω	

2b. Widerstände (Potentiometer)

	22.5 kΩ	(2)
	25 kΩ	(4)

3. LED’s:

	Rot	(11)

4. Dioden:

	1N4002	(8)
	1N4001	(9)

5. IC’s:

	DM7406N	(2)
	GD74LS74A	
	GD74LS273	
	7805C	
	L7812C-V	
	L7824C-V	

Anhang B

6. Transformatoren:

Ringkerntrafo 2x30V, 120W
Printtrafo 15V, 16W

7. Relais:

Hochlastrelais AC 220V, AC 220V, 30A
Schaltrelais DC 12V, AC 240V. (8)

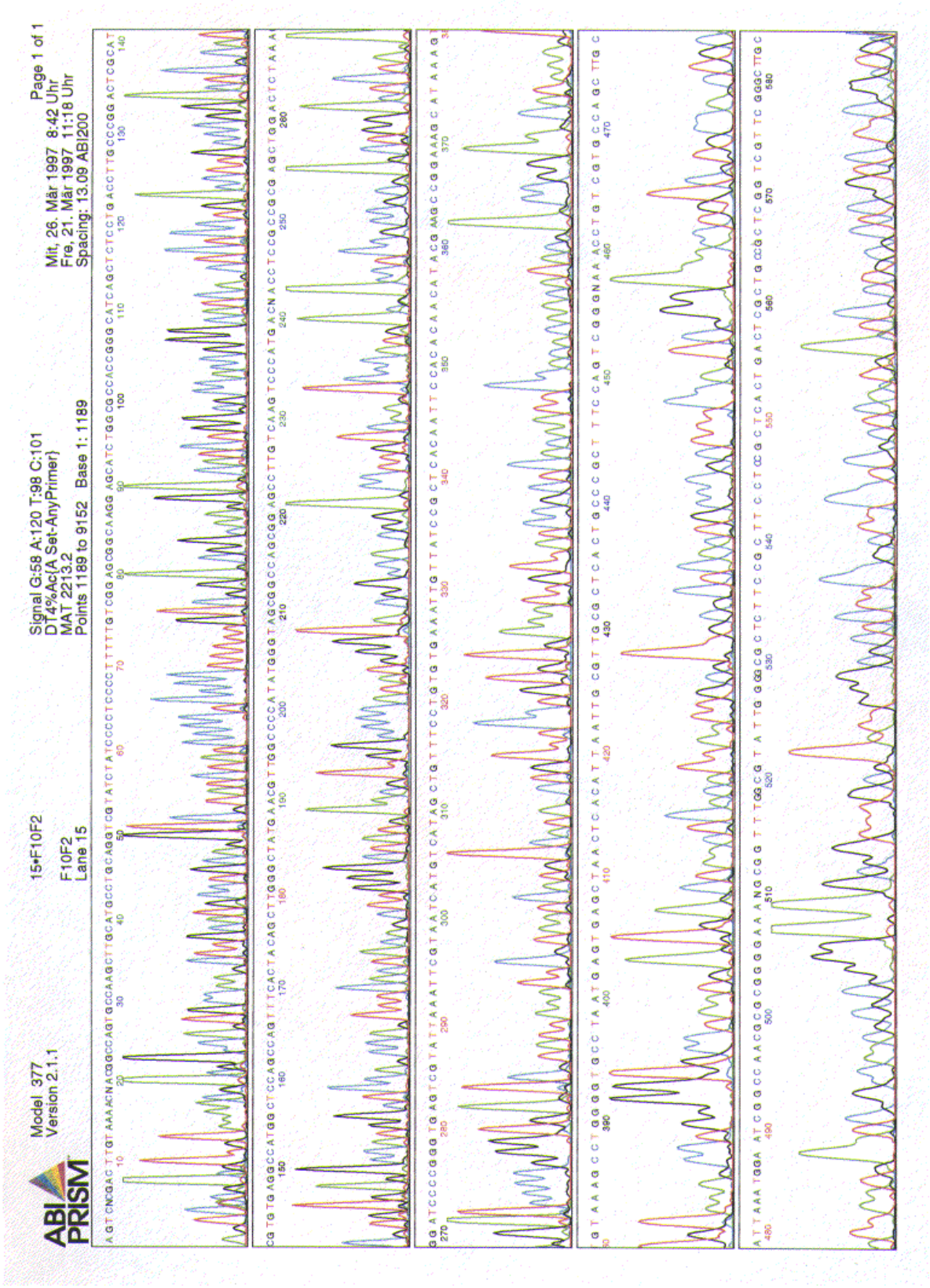
8. Anschlußbuchsen:

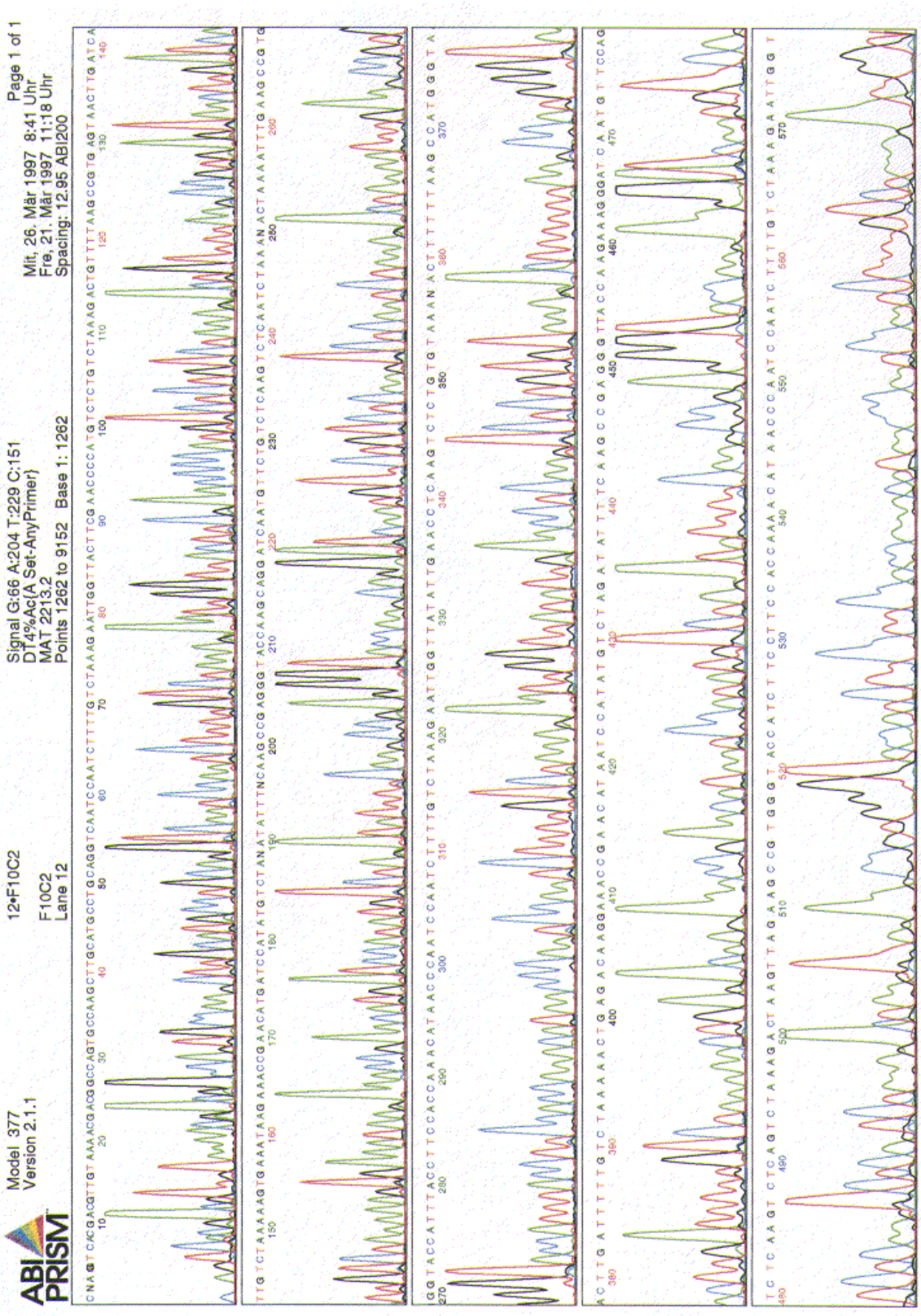
220V mit Schalter und Sicherungskasten
220V Einbaubuchse (2)
Niedervolteinbaubuchse 4 Polig (2)
Lüsterklemmen 4Polig (4)
Centronics Connector 36S

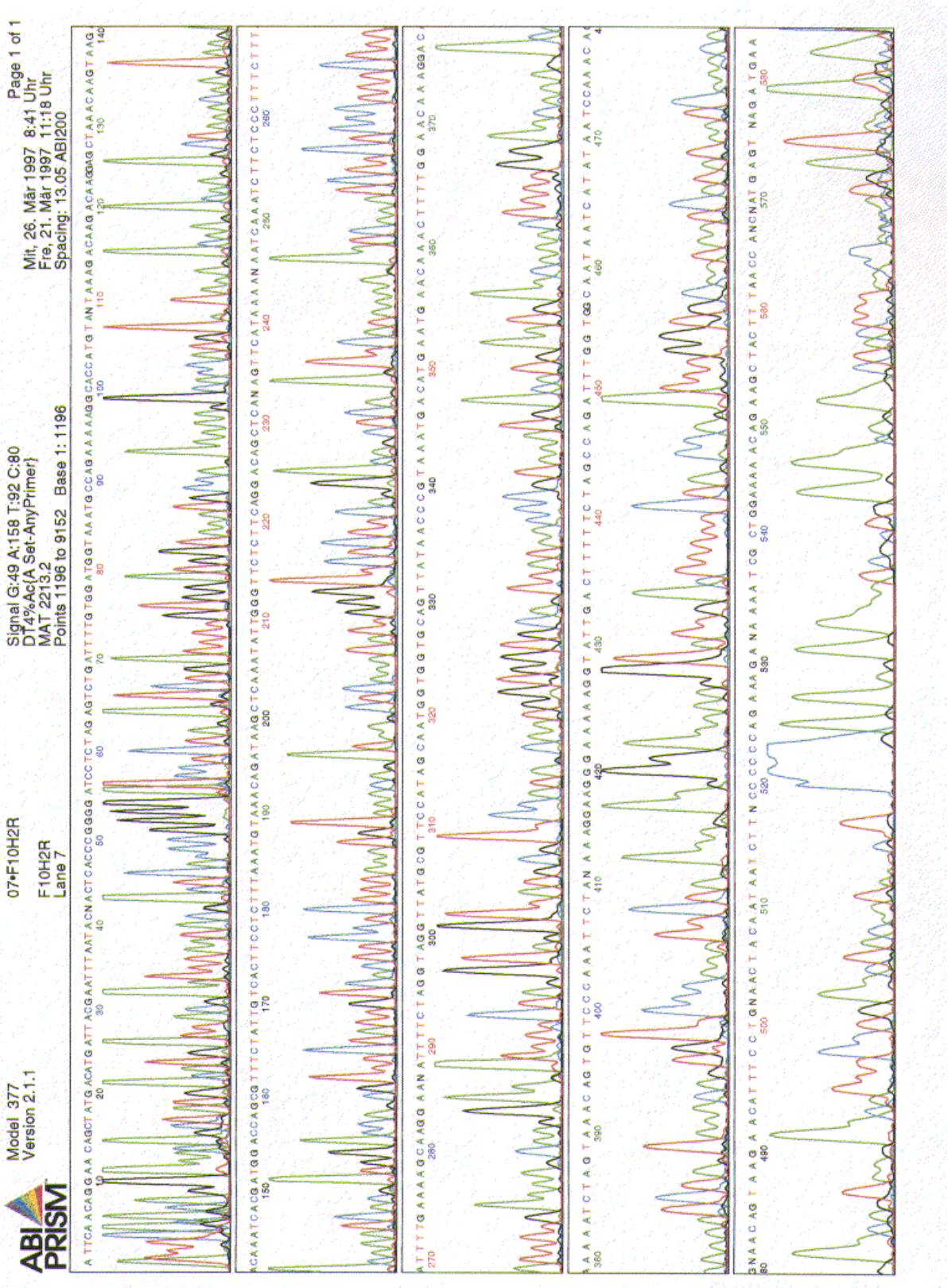
9. Sicherungen:

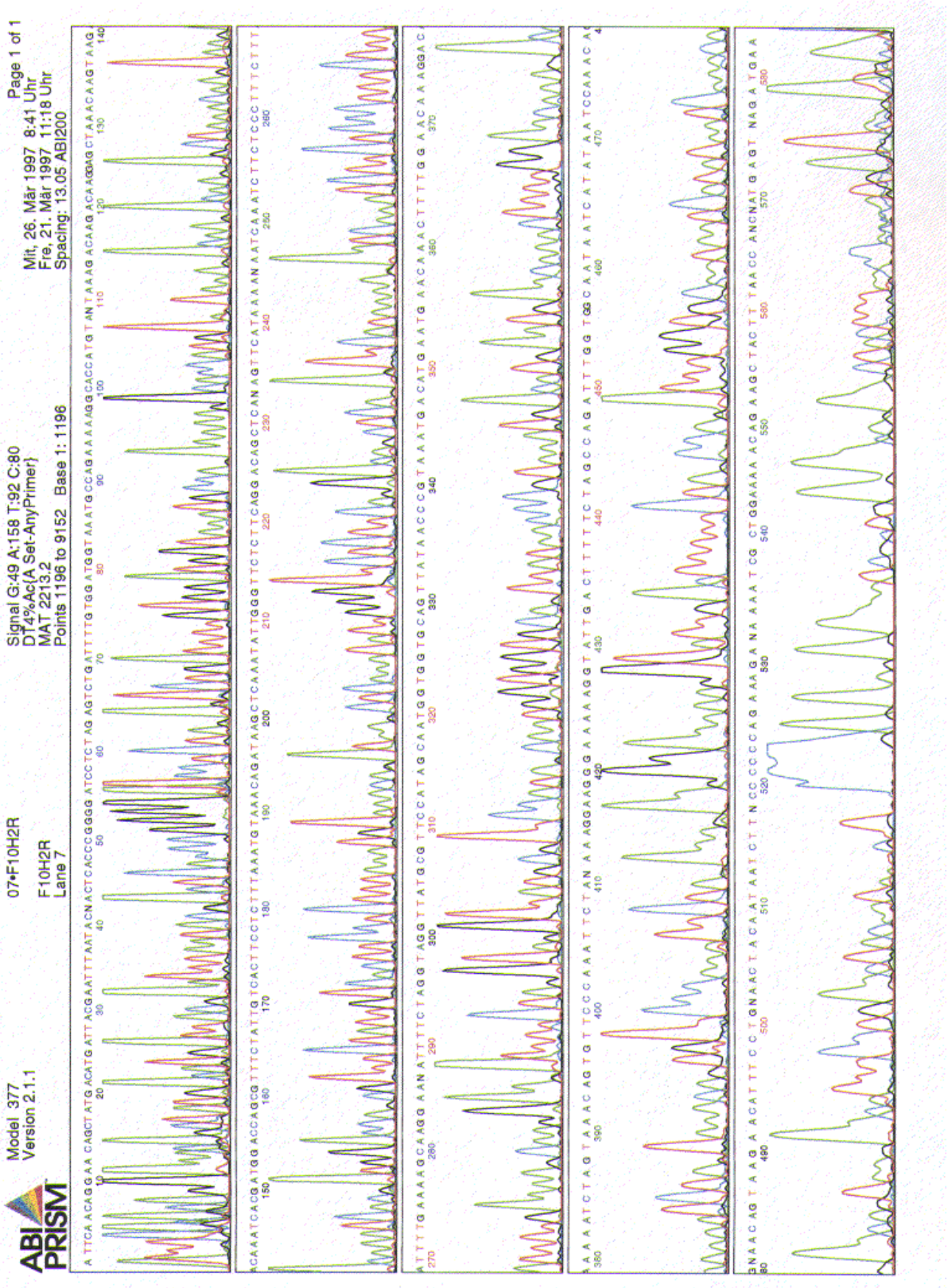
0.2A, flink.
2 A, flink

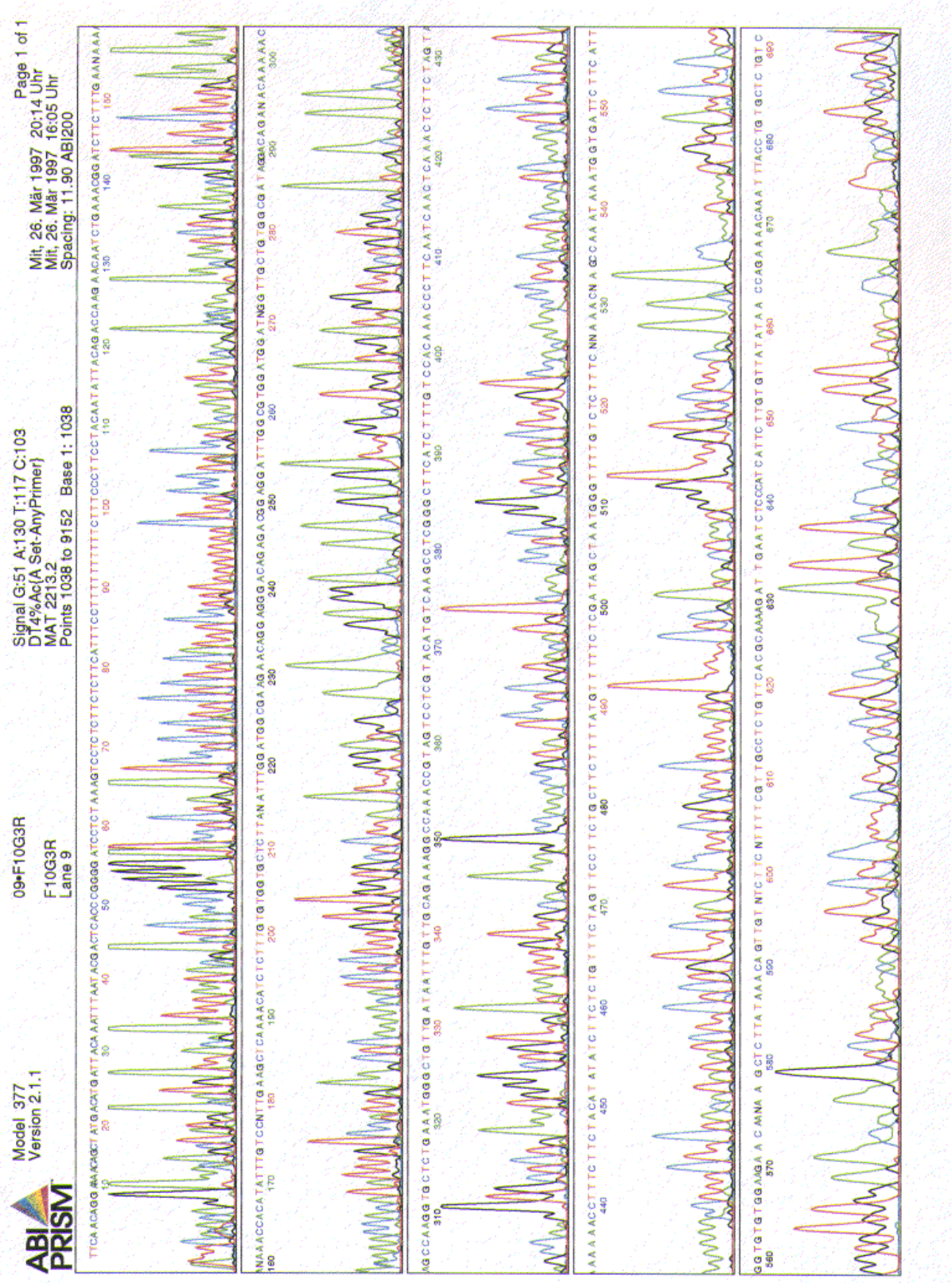
11 Anhang C: Elektropherogramme

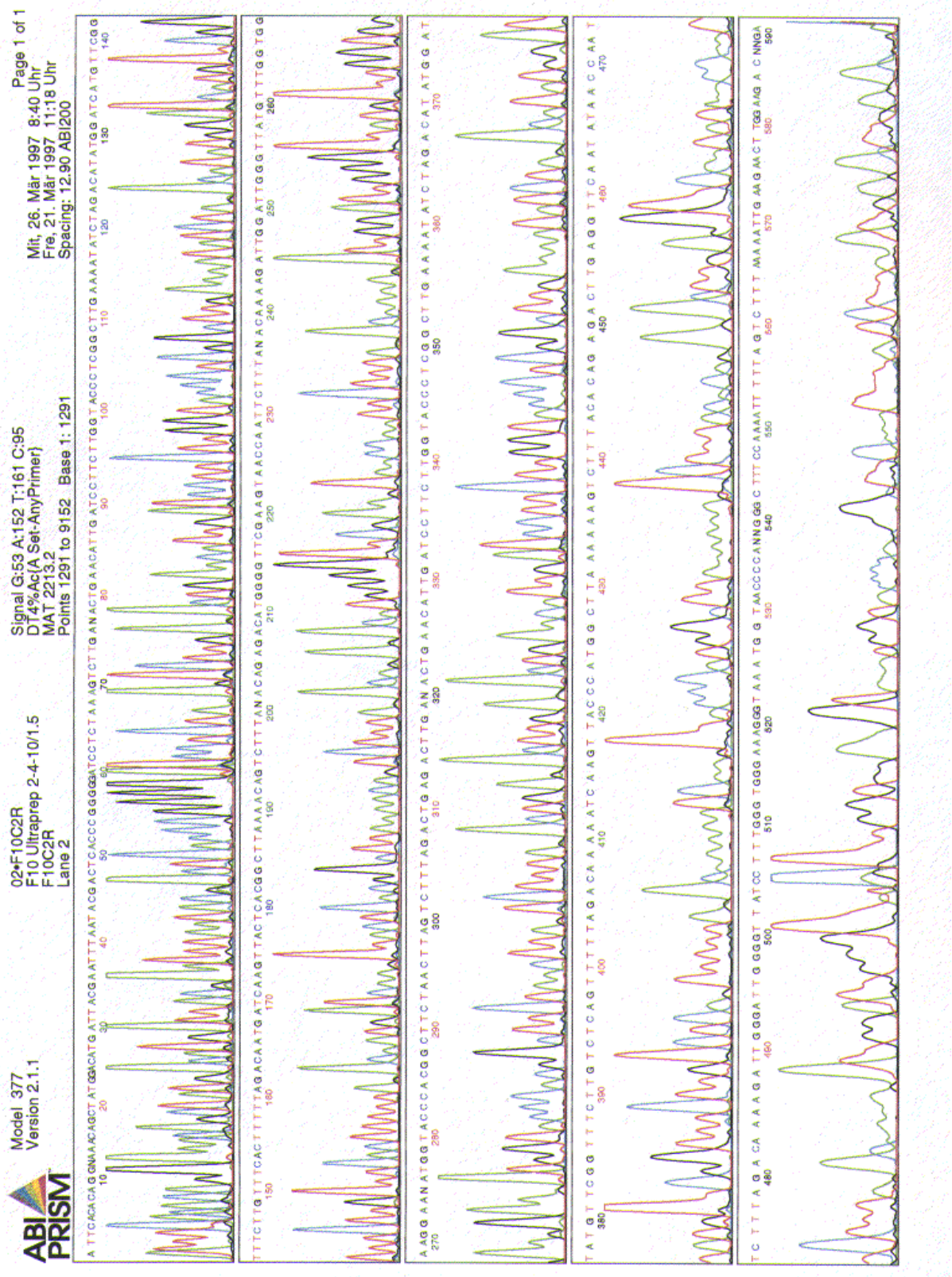


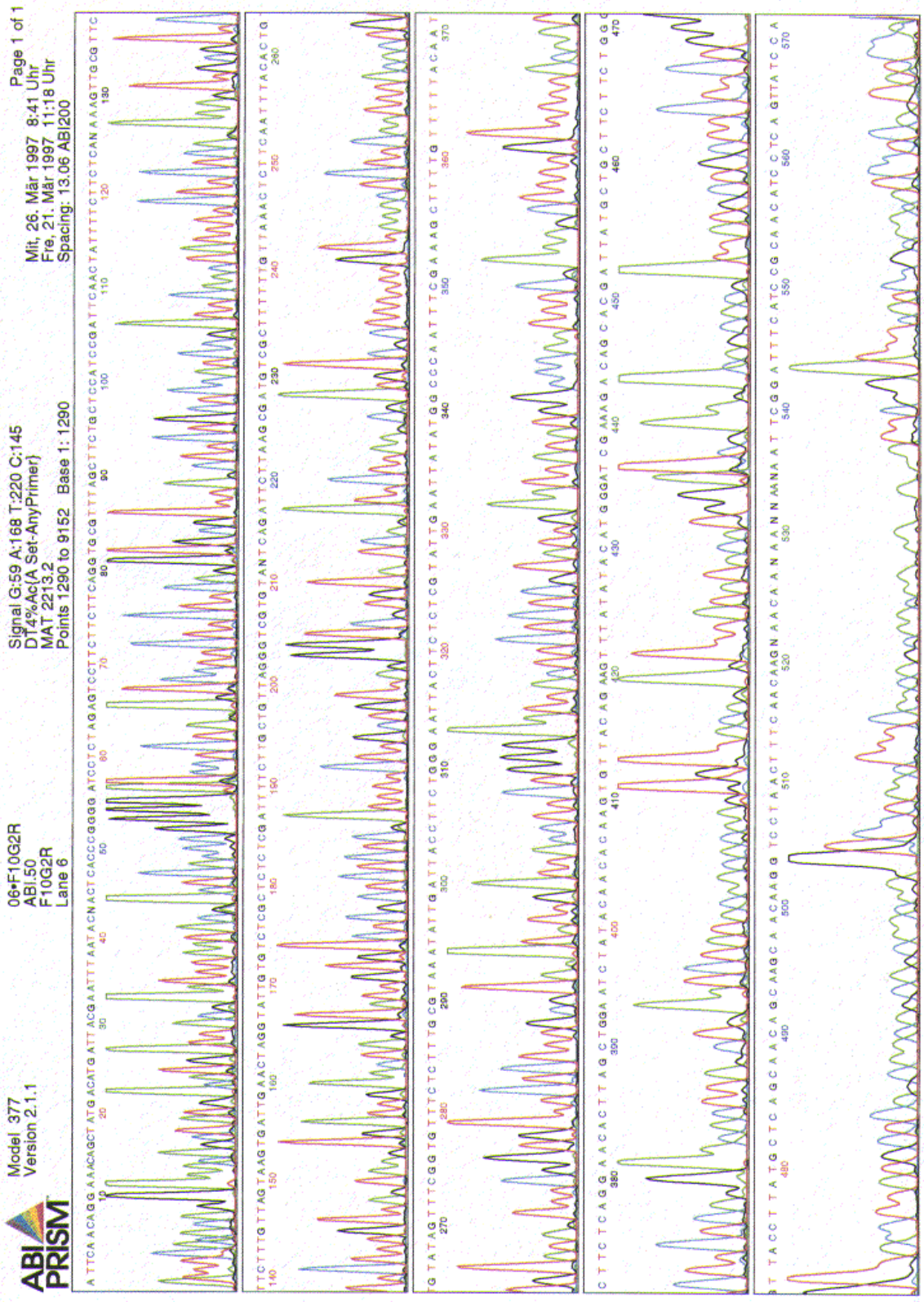

ABI PRISM
Model 377
Version 2.1.1
12•F10C2
F10C2
Lane 12
Signal G:66 A:204 T:229 C:151
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1262 to 9152 Base 1: 1262
Mit, 26. Mär 1997 8:41 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 12.95 ABI200
Page 1 of 1

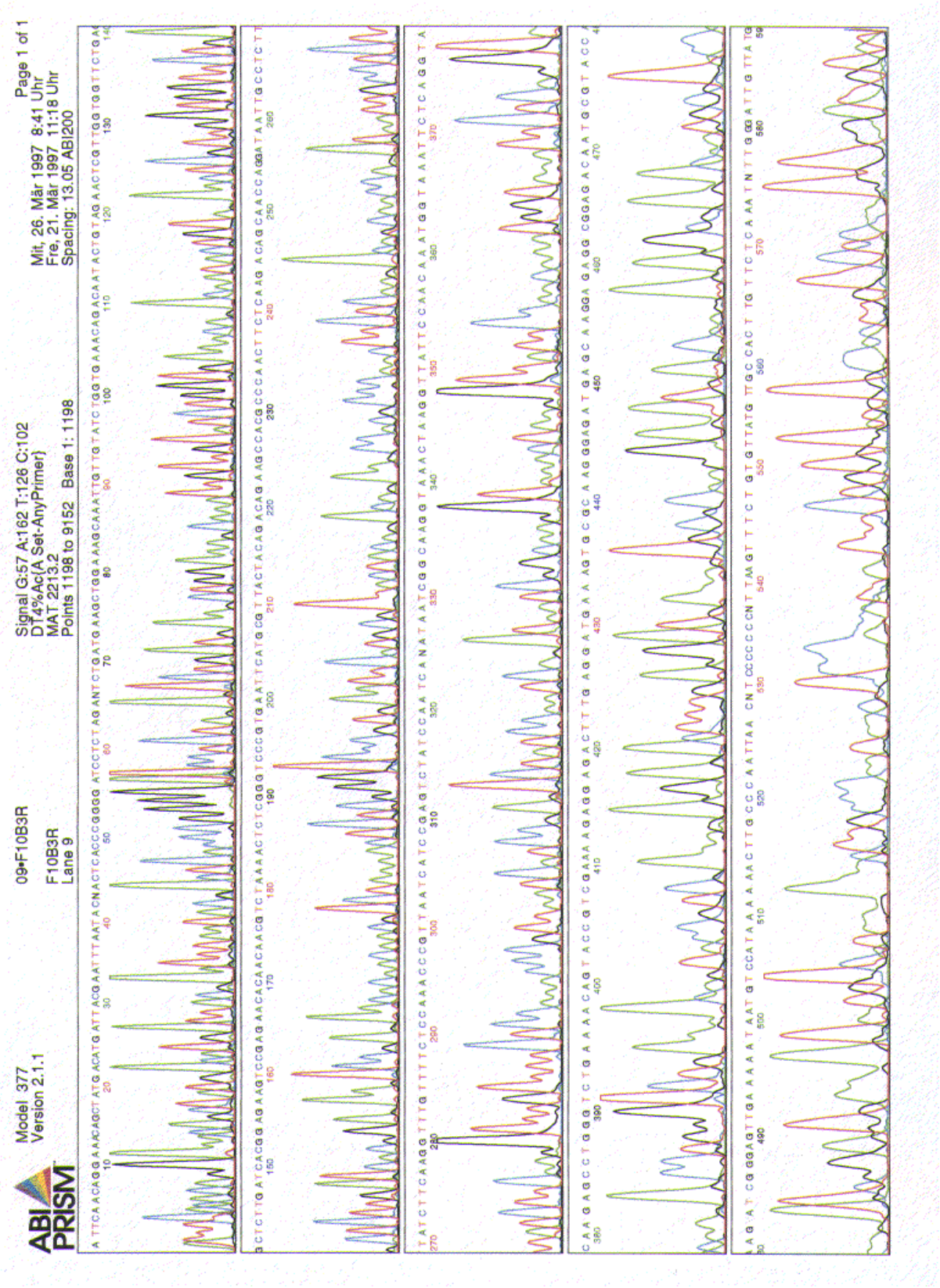

ABI PRISM
Model 377
Version 2.1.1
07•F10H2R
F10H2R
Lane 7
Signal G:49 A:158 T:92 C:80
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1196 to 9152 Base 1: 1196
Mit, 26. Mär 1997 8:41 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.05 ABI200
Page 1 of 1

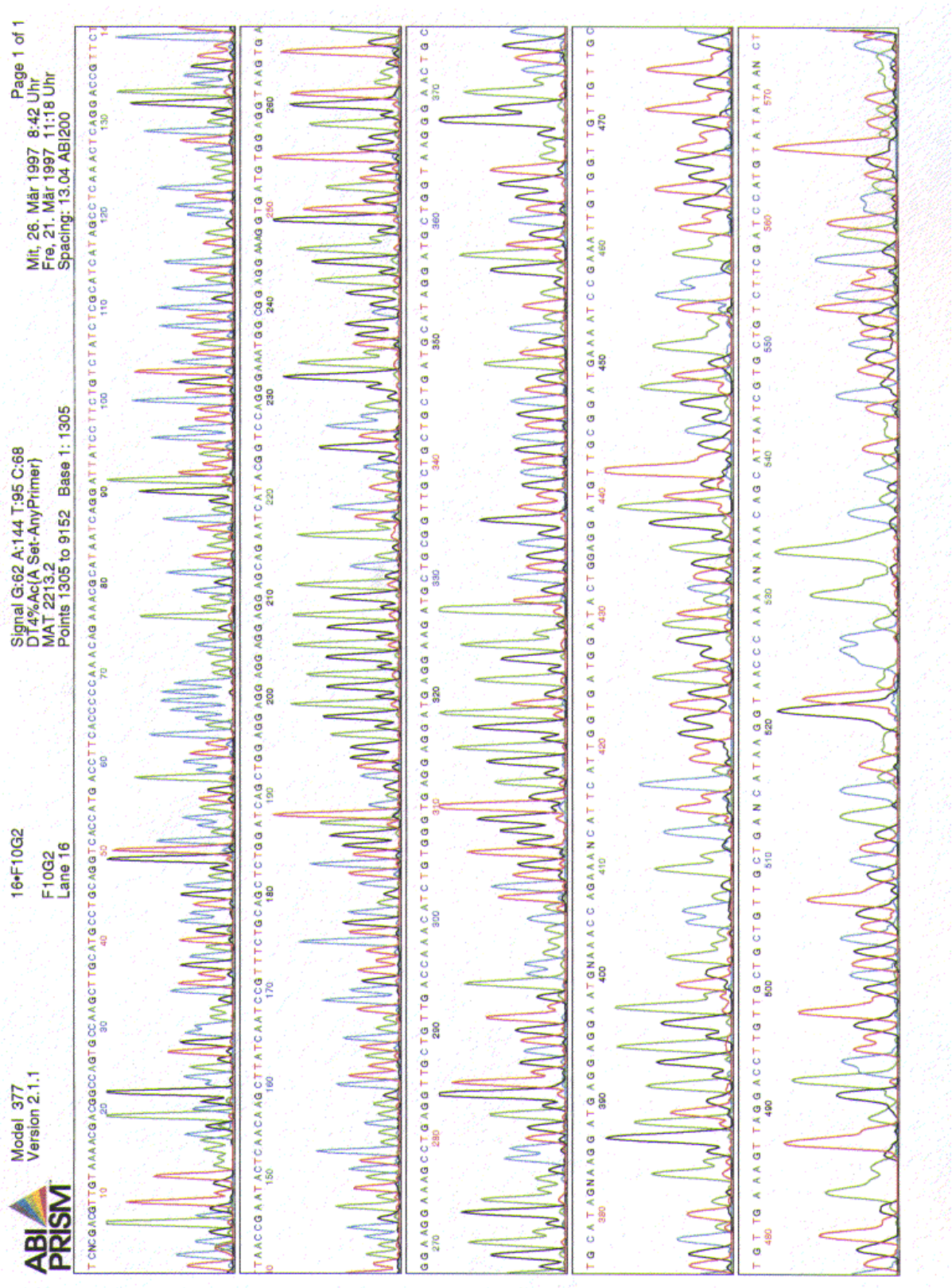

ABI PRISM
Model 377
Version 2.1.1
07•F10H2R
F10H2R
Lane 7
Signal G:49 A:158 T:92 C:80
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1196 to 9152 Base 1: 1196
Mit, 26. Mär 1997 8:41 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.05 ABI200
Page 1 of 1

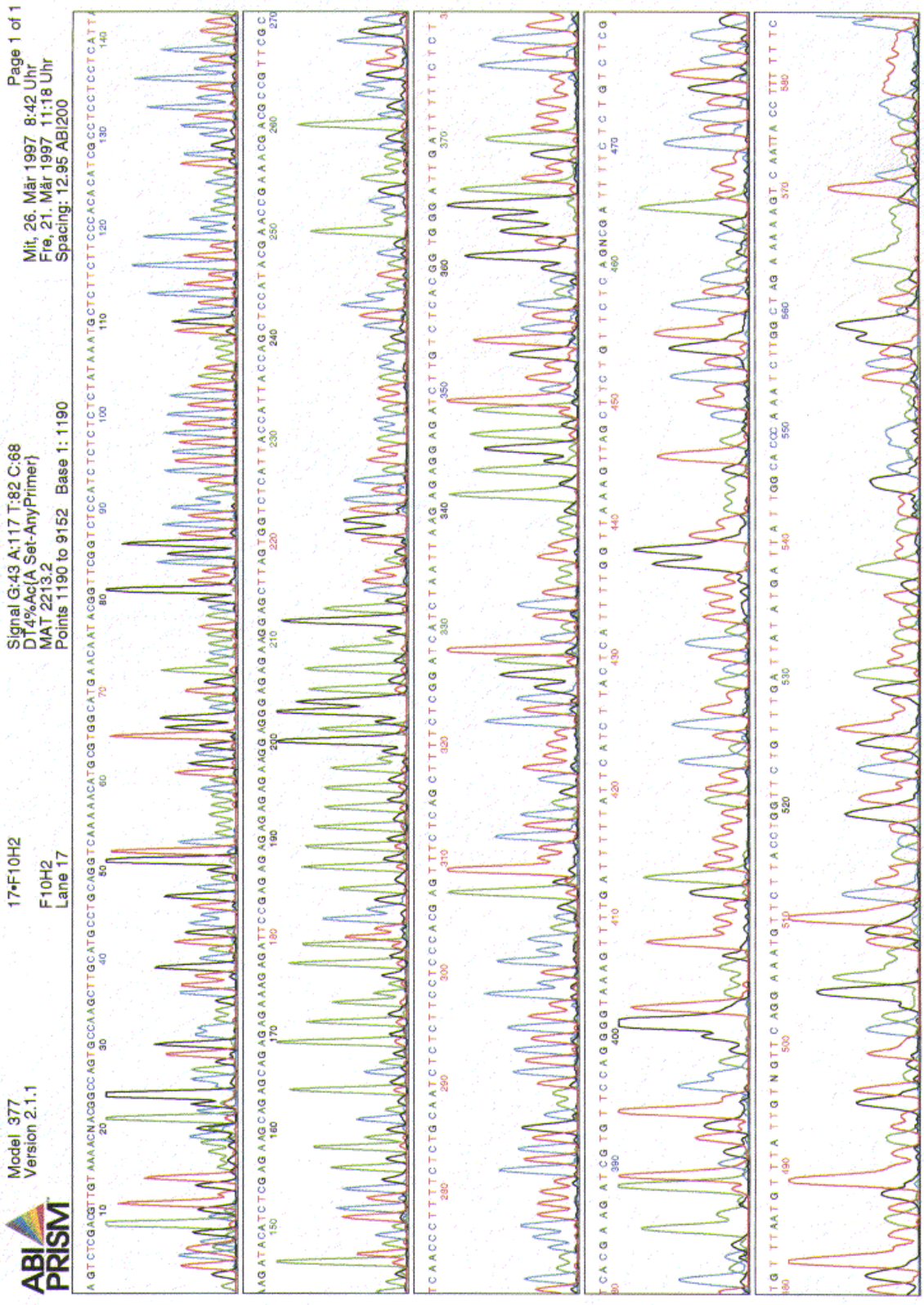

ABI PRISM
Model 377
Version 2.1.1
09•F10G3R
F10G3R
Lane 9
Signal G:51 A:130 T:117 C:103
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1038 to 9152 Base 1: 1038
Mit, 26. Mär 1997 20:14 Uhr
Mit, 26. Mär 1997 16:05 Uhr
Spacing: 11.90 ABI200
Page 1 of 1

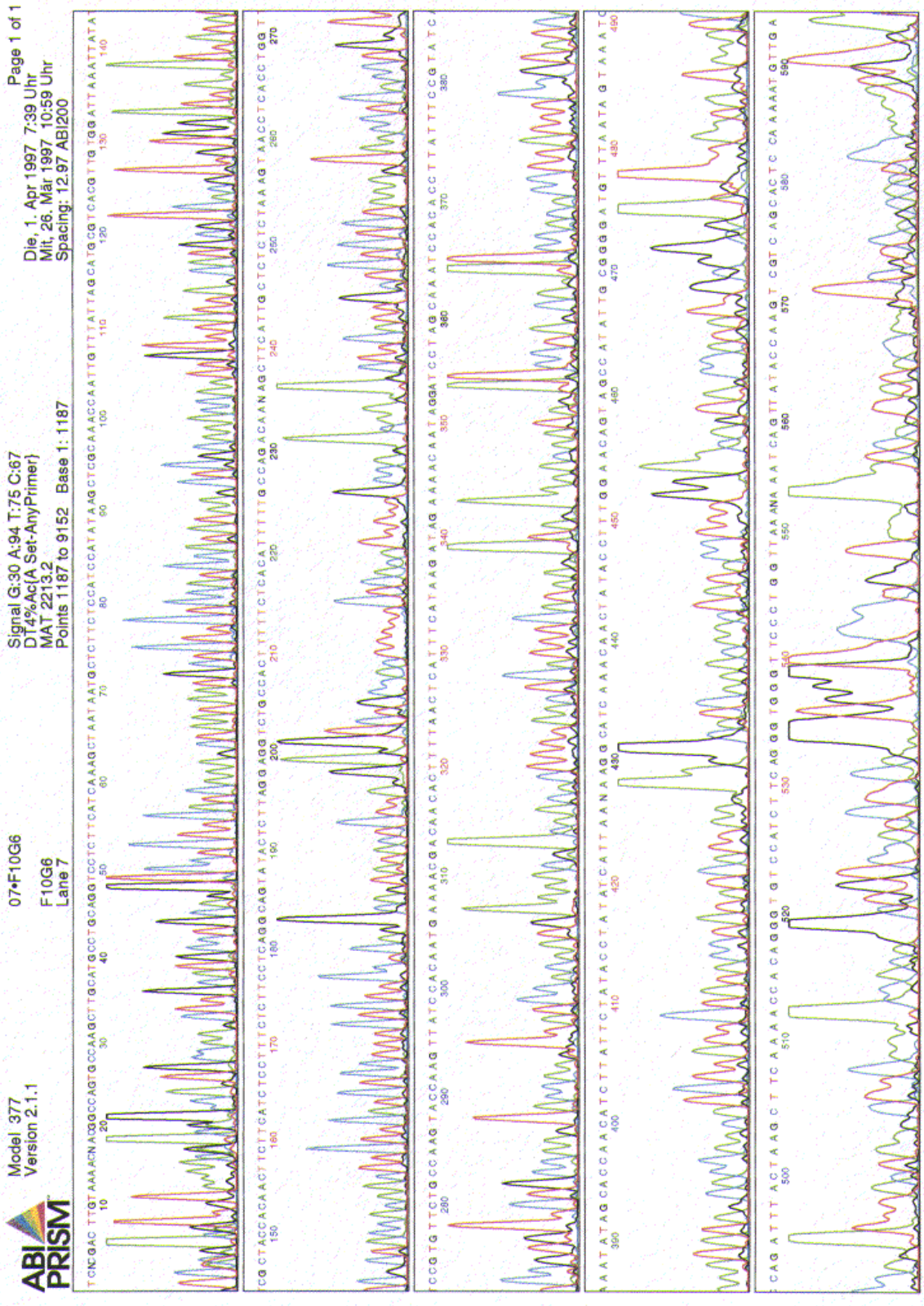

ABI PRISM
Model 377
Version 2.1.1
02rF10C2R
F10 Ultraprep 2-4-10/1.5
F10C2R
Lane 2
Signal G:53 A:152 T:161 C:95
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1291 to 9152 Base 1: 1291
Mit, 26. Mär 1997 8:40 Uhr
Frè, 21. Mär 1997 11:18 Uhr
Spacing: 12.90 ABI200
Page 1 of 1

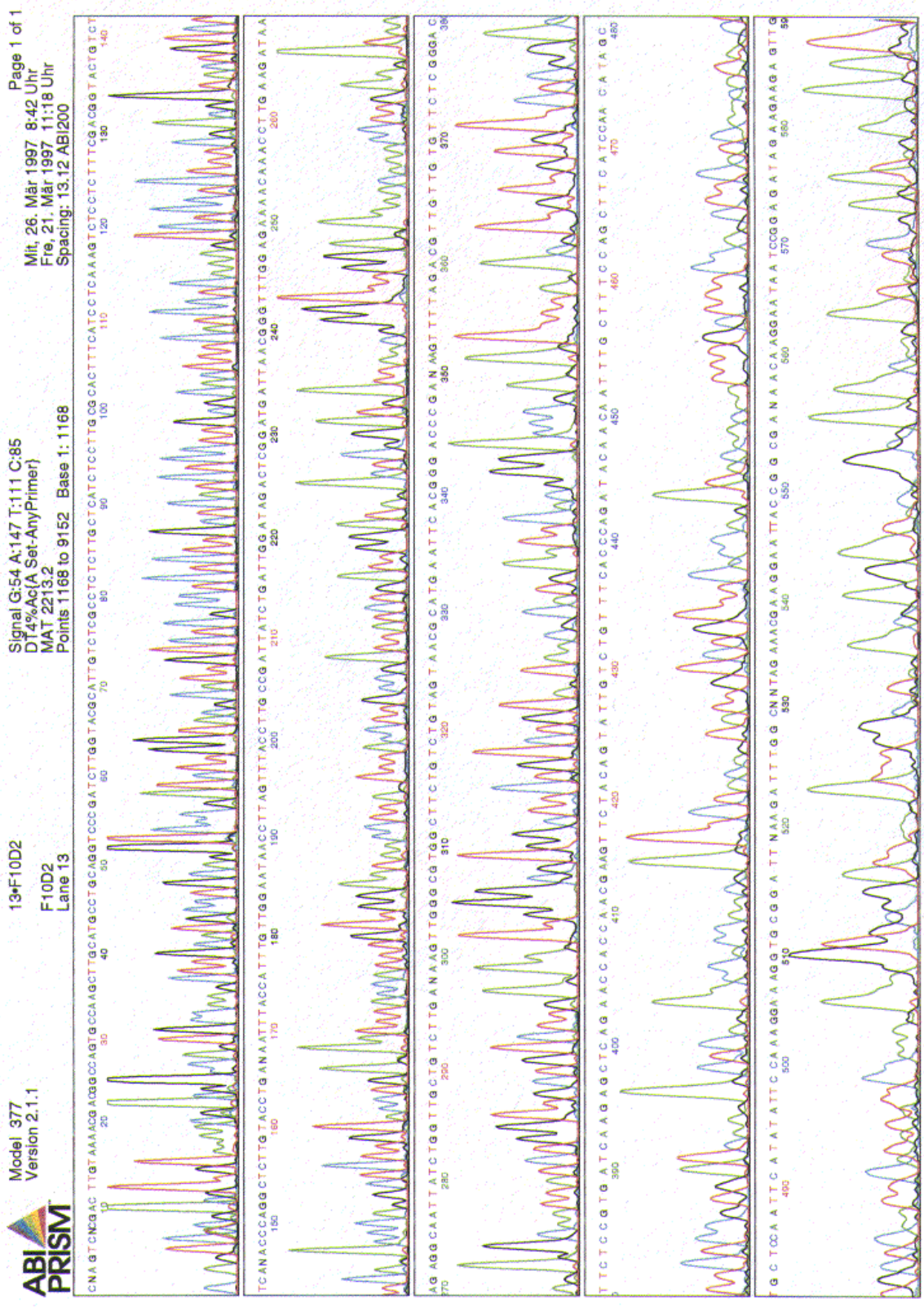

ABI PRISM
Model 377
Version 2.1.1
06+F10G2R
ABI.50
F10G2R
Lane 6
Signal G:59 A:168 T:220 C:145
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1290 to 9152 Base 1: 1290
Mit, 26. Mär 1997 8:41 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.06 ABI200
Page 1 of 1

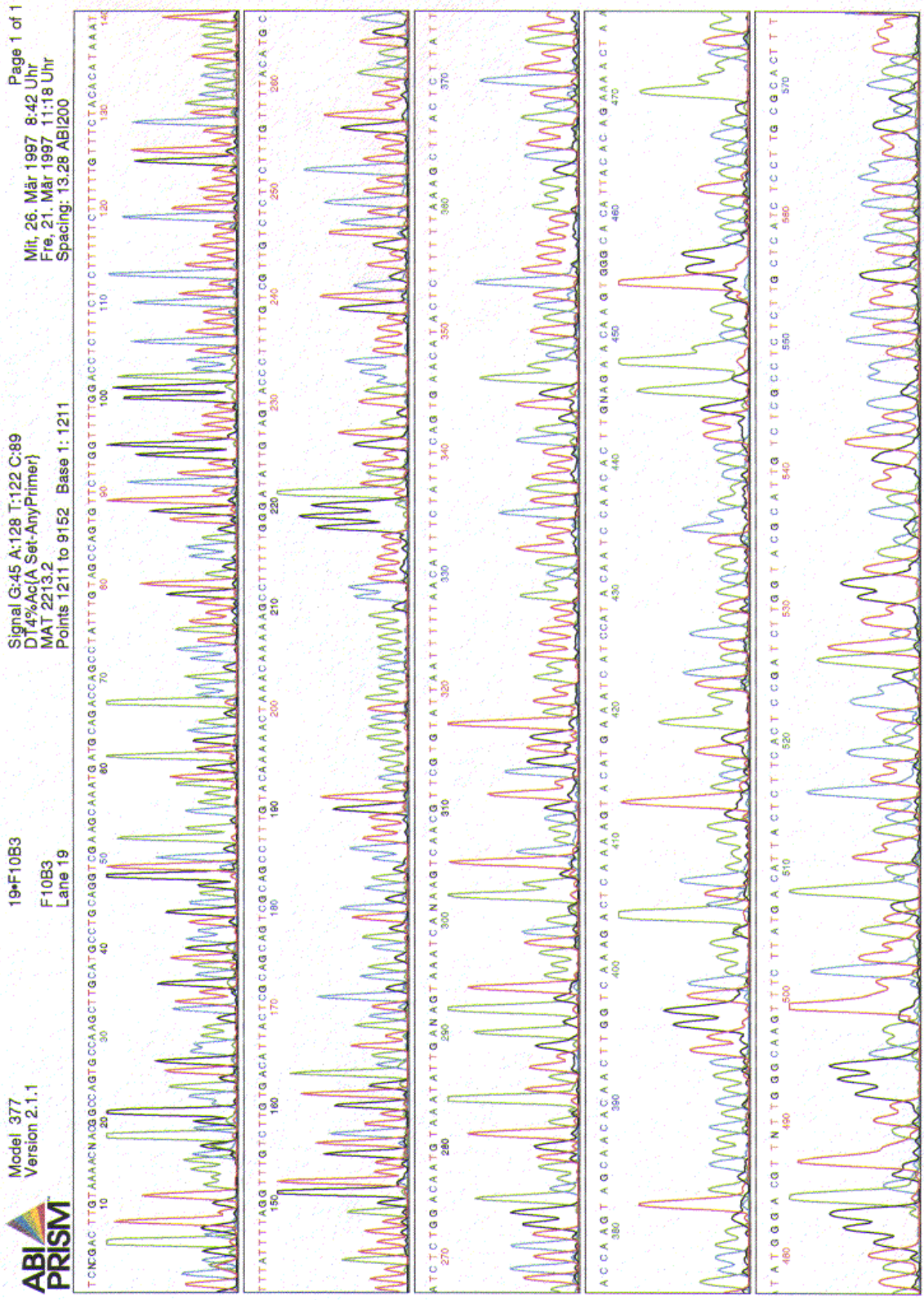

ABI PRISM
Model 377
Version 2.1.1
09•F10B3R
F10B3R
Lane 9
Signal G:57 A:162 T:126 C:102
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1198 to 9152 Base 1: 1198
Mit, 26. Mär 1997 8:41 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.05 ABI200
Page 1 of 1

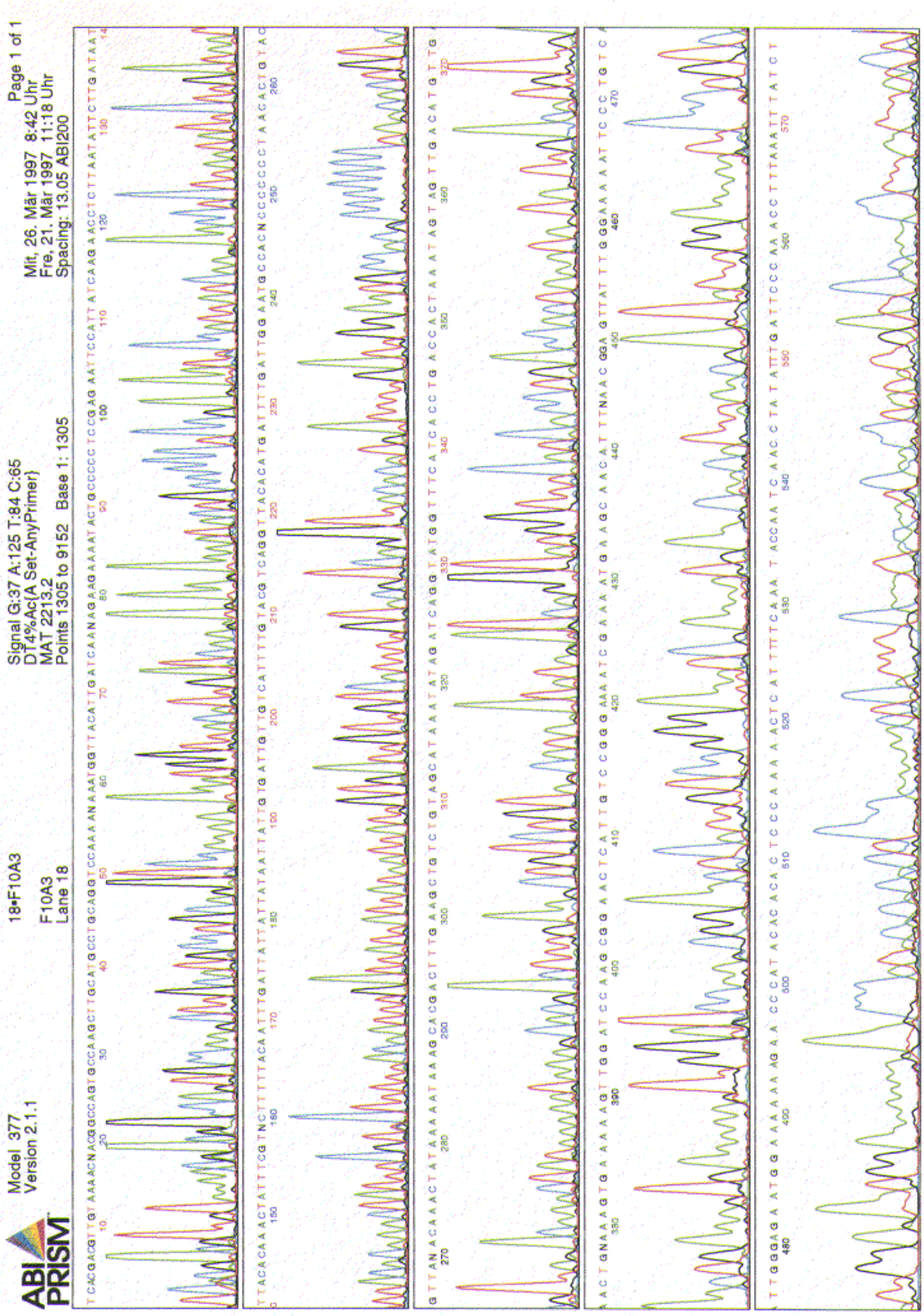

ABI PRISM
Model 377
Version 2.1.1
16rF10G2
F10G2
Lane 16
Signal G:62 A:144 T:95 C:68
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1305 to 9152 Base 1: 1305
Mit, 26. Mär 1997 8:42 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.04 ABI200
Page 1 of 1

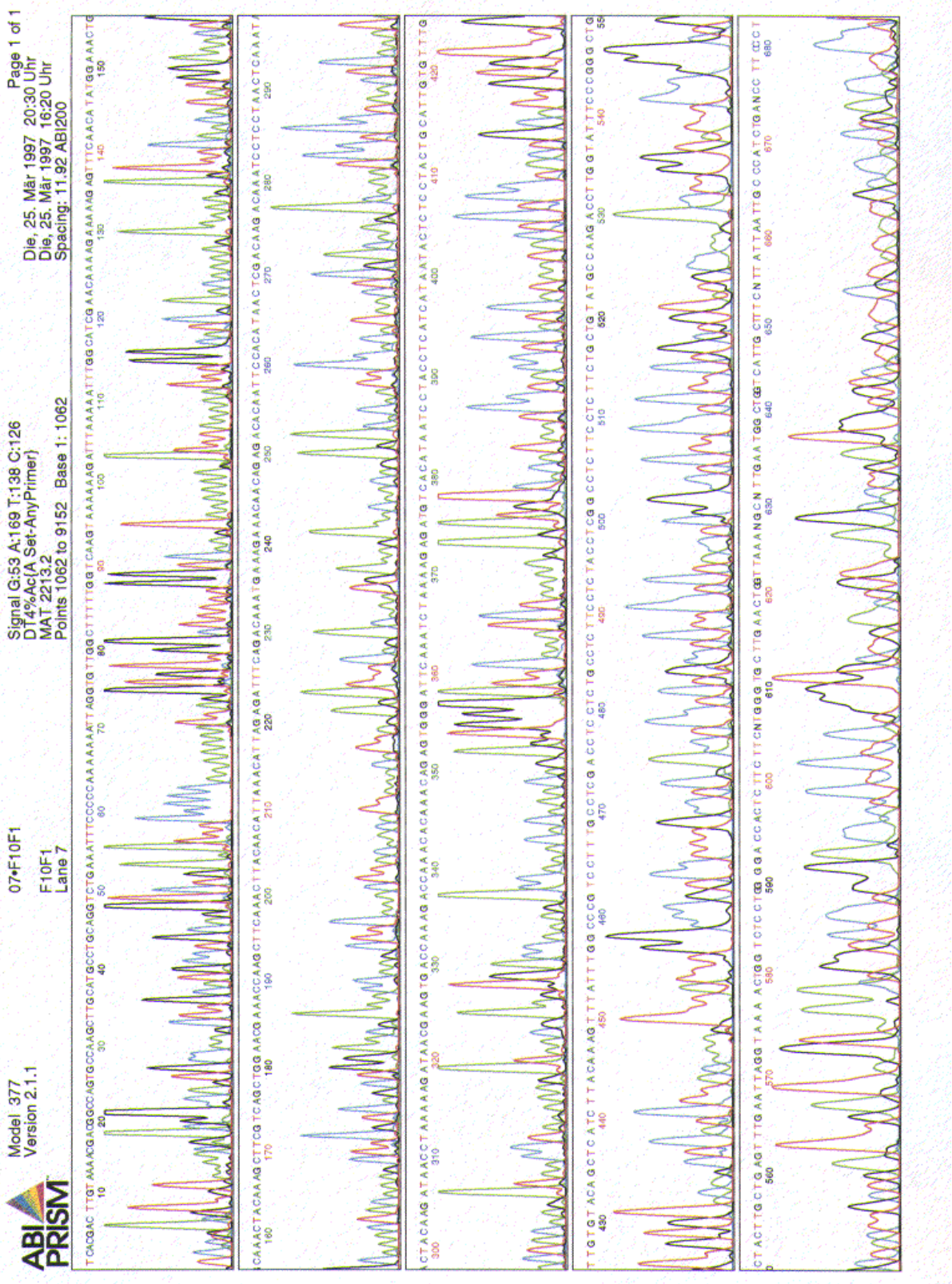

ABI PRISM
Model 377
Version 2.1.1
17•F10H2
F10H2
Lane 17
Signal G:43 A:117 T:82 C:68
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1190 to 9152 Base 1: 1190
Mit, 26. Mär 1997 8:42 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 12.95 ABI200
Page 1 of 1

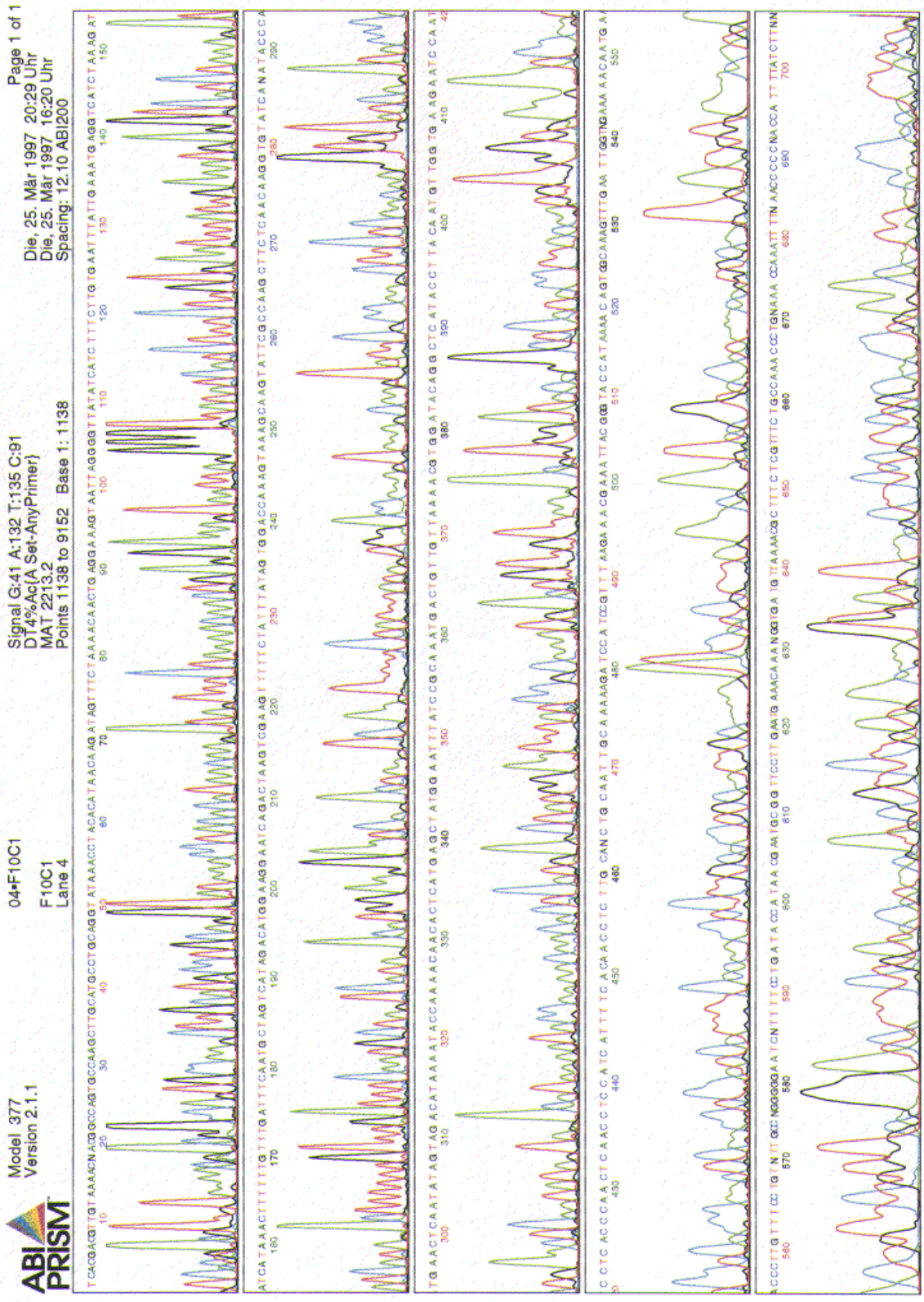

ABI PRISM
Model 377
Version 2.1.1
07•F10G6
F10G6
Lane 7
Signal G:30 A:94 T:75 C:67
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1187 to 9152 Base 1: 1187
Die, 1. Apr 1997 7:39 Uhr
Mit, 26. Mär 1997 10:59 Uhr
Spacing: 12.97 ABI200
Page 1 of 1

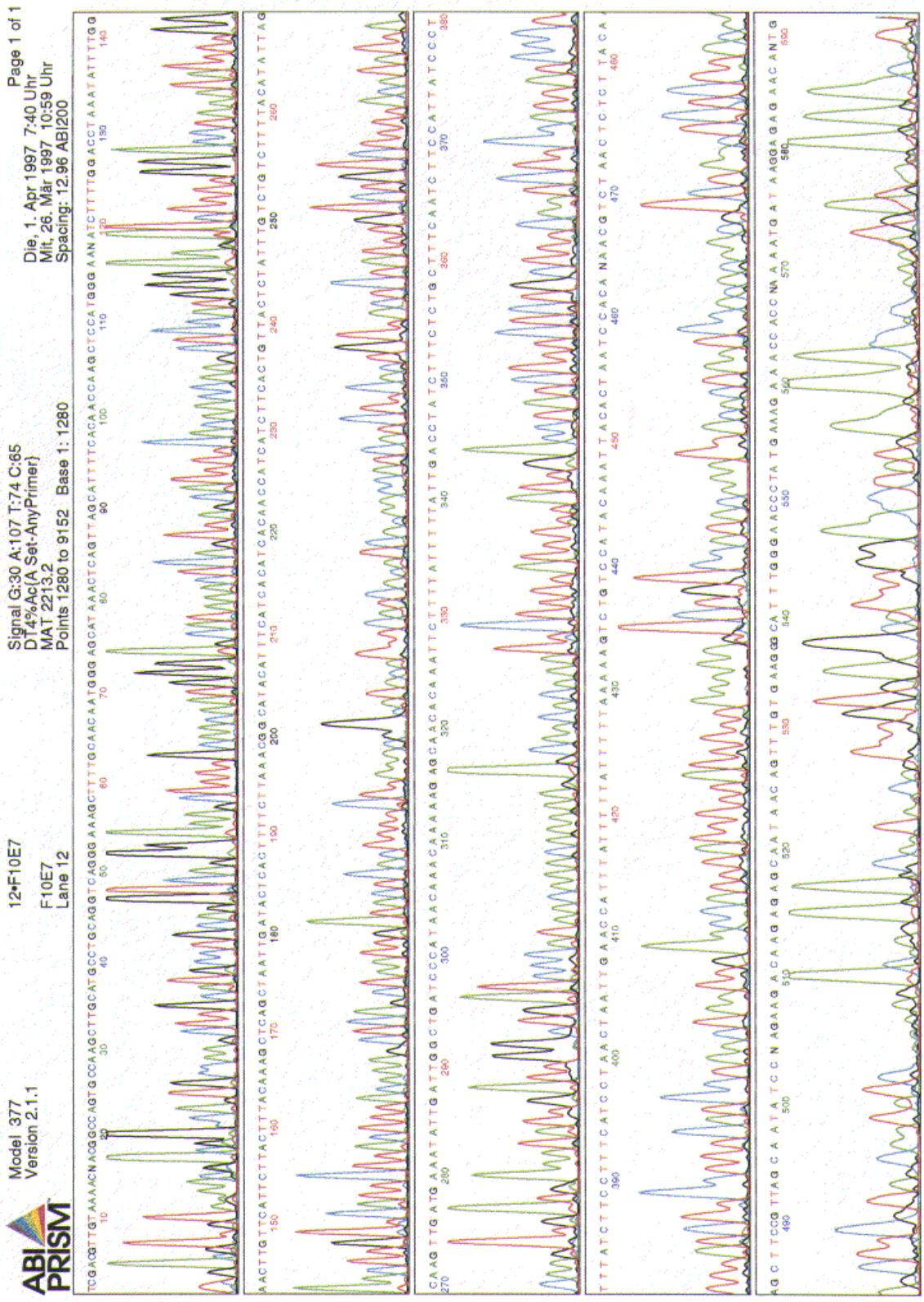

ABI PRISM
Model 377
Version 2.1.1
13•F10D2
F10D2
Lane 13
Signal G:54 A:147 T:111 C:85
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1168 to 9152 Base 1: 1168
Mit, 26. Mär 1997 8:42 Uhr
Frę, 21. Mär 1997 11:18 Uhr
Spacing: 13.12 ABI200
Page 1 of 1

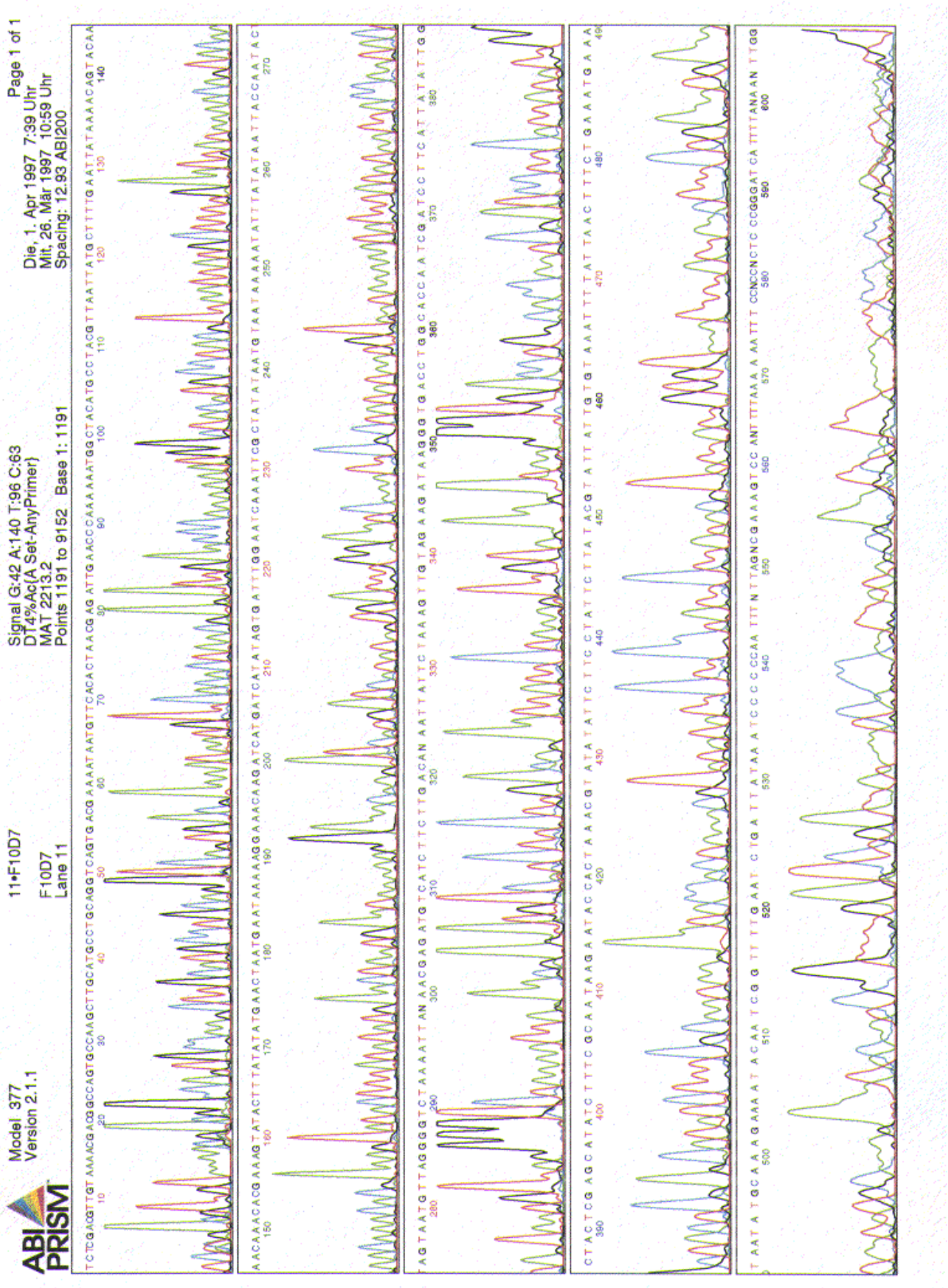

ABI PRISM
Model 377
Version 2.1.1
19•F10B3
F10B3
Lane 19
Signal G:45 A:128 T:122 C:89
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1211 to 9152 Base 1: 1211
Mit, 26. Mär 1997 8:42 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.28 ABI200
Page 1 of 1

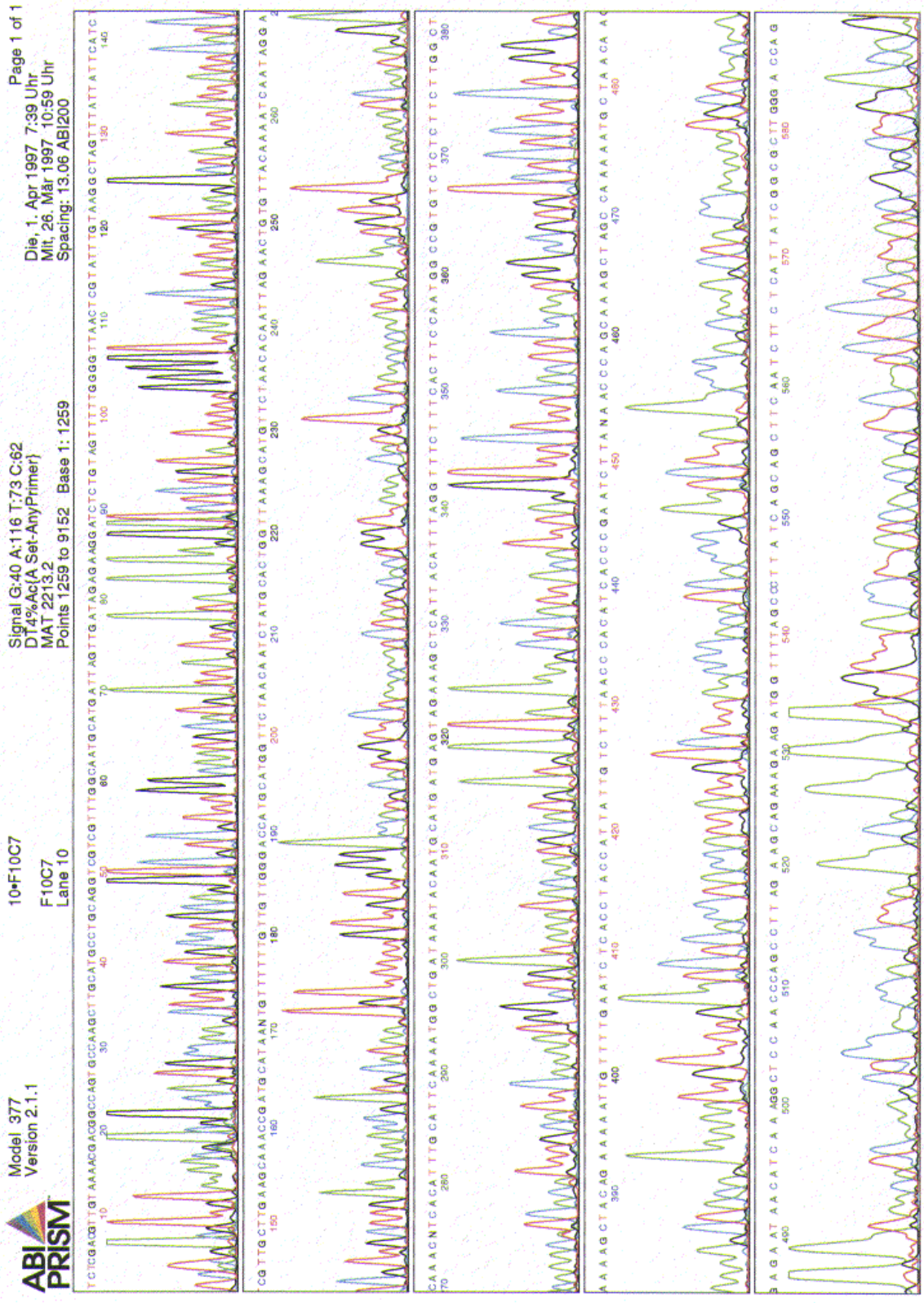

ABI PRISM
Model 377
Version 2.1.1
18•F10A3
F10A3
Lane 18
Signal G:37 A:125 T:84 C:65
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1305 to 9152 Base 1: 1305
Mit, 26. Mär 1997 8:42 Uhr
Fre, 21. Mär 1997 11:18 Uhr
Spacing: 13.05 ABI200
Page 1 of 1

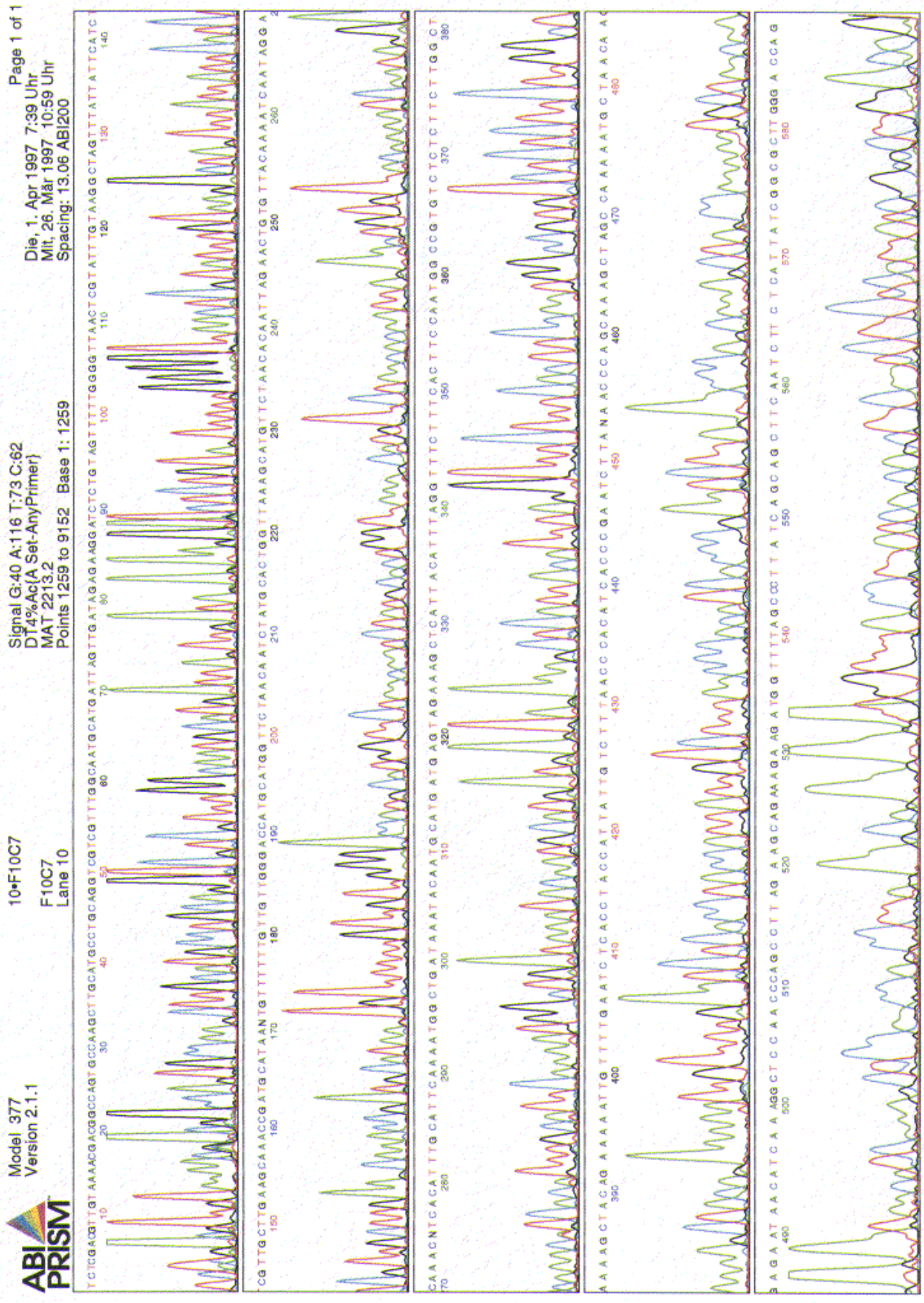

ABI PRISM
Model 377
Version 2.1.1
07•F10F1
F10F1
Lane 7
Signal G:53 A:169 T:138 C:126
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1062 to 9152 Base 1: 1062
Die, 25. Mär 1997 20:30 Uhr
Die, 25. Mär 1997 16:20 Uhr
Spacing: 11.92 ABI200
Page 1 of 1

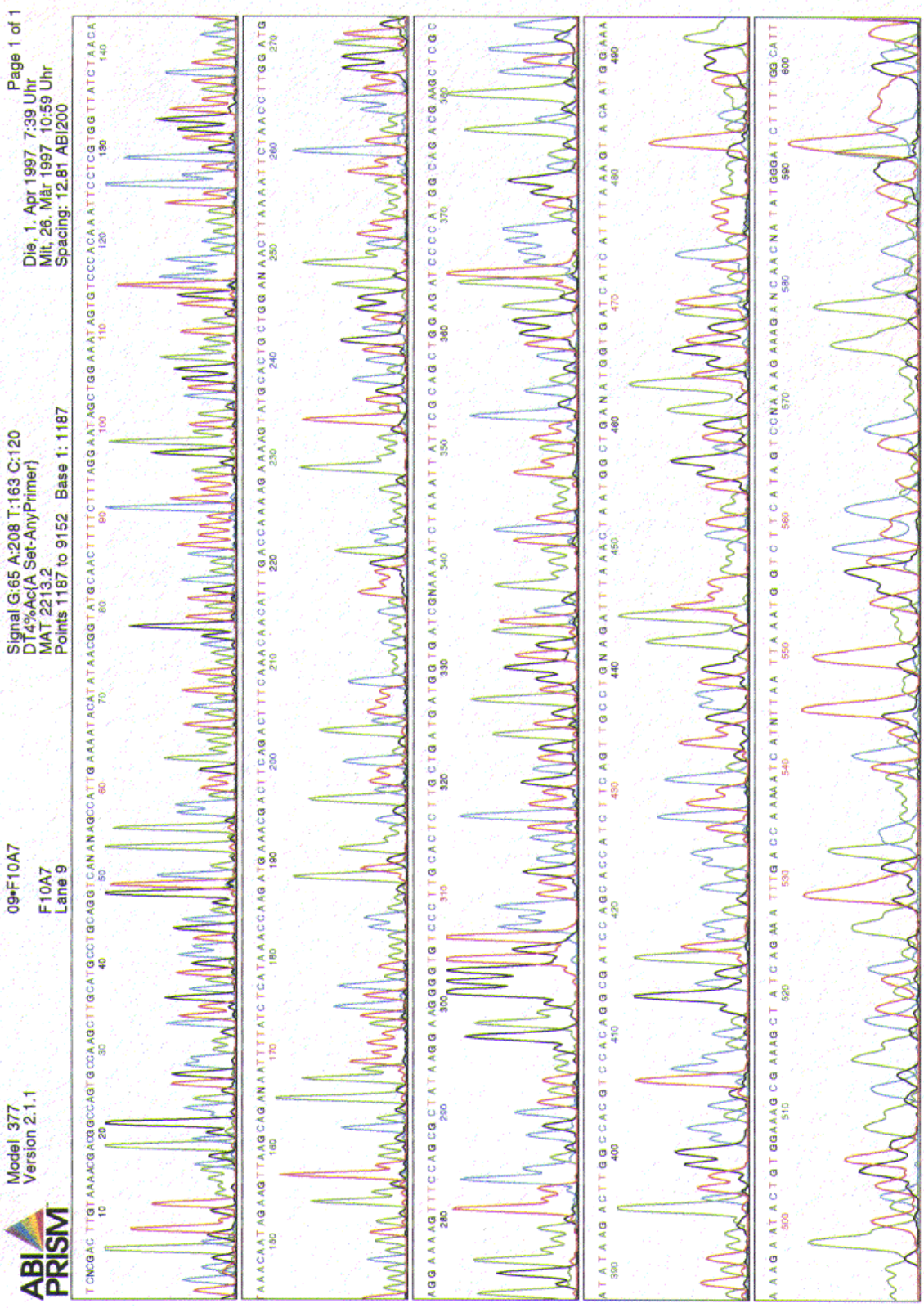

ABI PRISM
Model 377
Version 2.1.1
04•F10C1
F10C1
Lane 4
Signal G:41 A:132 T:135 C:91
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1138 to 9152 Base 1: 1138
Die, 25. Mär 1997 20:29 Uhr
Die, 25. Mär 1997 16:20 Uhr
Spacing: 12.10 ABI200
Page 1 of 1

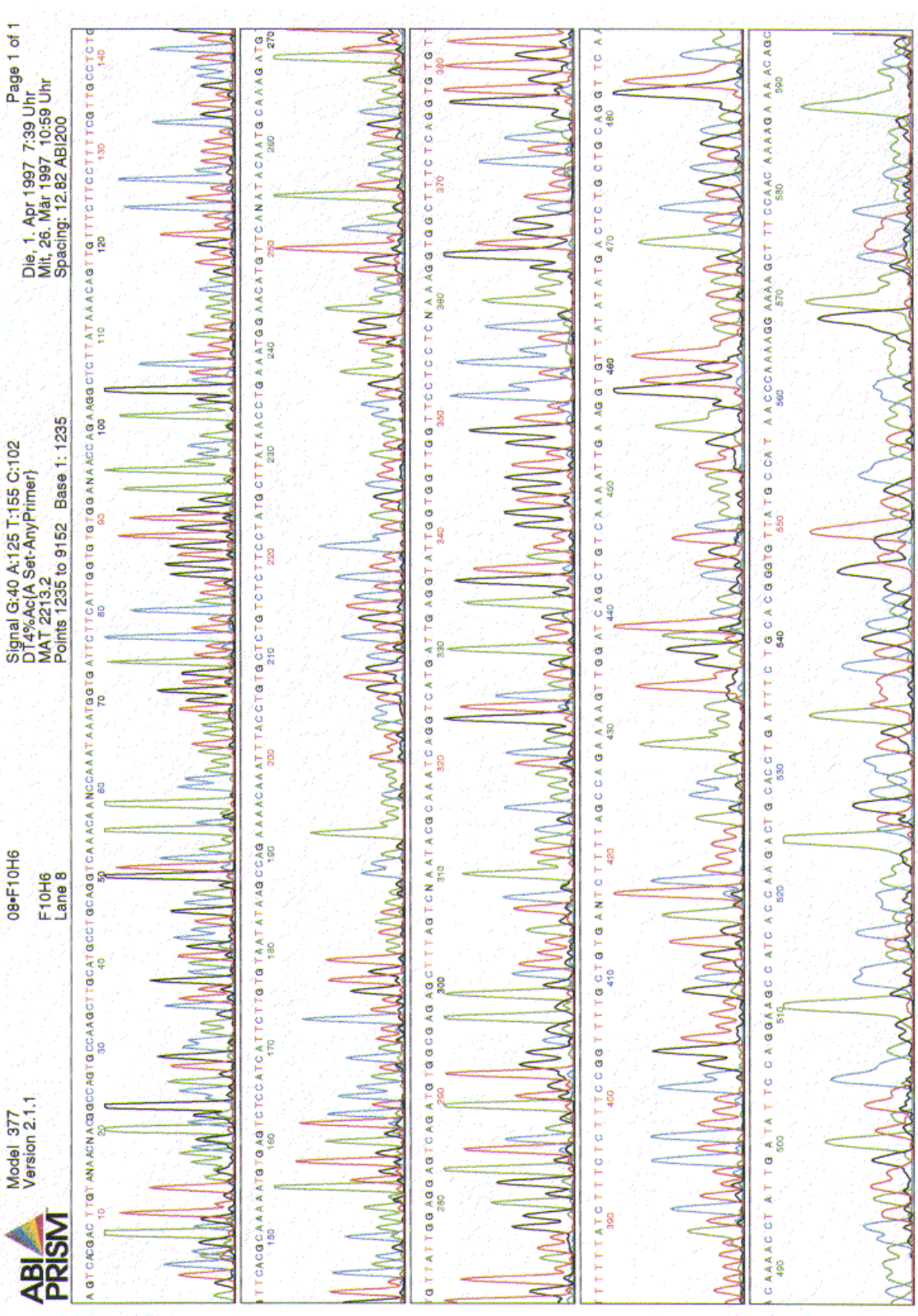

ABI PRISM
Model 377
Version 2.1.1
12•F10E7
F10E7
Lane 12
Signal G:30 A:107 T:74 C:65
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1280 to 9152 Base 1: 1280
Die, 1. Apr 1997 7:40 Uhr
Mit, 26. Mär 1997 10:59 Uhr
Spacing: 12.96 ABI200
Page 1 of 1


ABI PRISM
Model 377
Version 2.1.1
11•F10D7
F10D7
Lane 11
Signal G:42 A:140 T:96 C:63
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1191 to 9152 Base 1: 1191
Die, 1. Apr 1997 7:39 Uhr
Mit, 26. Mar 1997 10:59 Uhr
Spacing: 12.93 ABI200
Page 1 of 1


ABI PRISM
Model 377
Version 2.1.1
10•F10C7
F10C7
Lane 10
Signal G:40 A:116 T:73 C:62
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1259 to 9152 Base 1: 1259
Die, 1. Apr 1997 7:39 Uhr
Mit, 26. Mär 1997 10:59 Uhr
Spacing: 13.06 ABI200
Page 1 of 1


ABI PRISM
Model 377
Version 2.1.1
10•F10C7
F10C7
Lane 10
Signal G:40 A:116 T:73 C:62
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1259 to 9152 Base 1: 1259
Die, 1. Apr 1997 7:39 Uhr
Mit, 26. Mär 1997 10:59 Uhr
Spacing: 13.06 ABI200
Page 1 of 1


ABI PRISM
Model 377
Version 2.1.1
09•F10A7
F10A7
Lane 9
Signal G:65 A:208 T:163 C:120
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1187 to 9152 Base 1: 1187
Die, 1. Apr 1997 7:39 Uhr
Mit, 26. Mär 1997 10:59 Uhr
Spacing: 12.81 ABI200
Page 1 of 1


ABI PRISM
Model 377
Version 2.1.1
08•F10H6
F10H6
Lane 8
Signal G:40 A:125 T:155 C:102
DT4%Ac(A Set-AnyPrimer)
MAT 2213.2
Points 1235 to 9152 Base 1: 1235
Die, 1. Apr 1997 7:39 Uhr
Mit, 26. Mär 1997 10:59 Uhr
Spacing: 12.82 ABI200
Page 1 of 1